InnovaCampus 2.0:
Redisseny d'una aplicació web d'ajut a l'autoavaluació formativa

Xavier Aiguabella Guilera, Natàlia Sans Solsona

2009

Abstract

InnovaCampus 2.0 is the redesigning of the free software application InnovaCampus using the frameworks Spring and JPA.

InnovaCampus is an application that allows university students test themselves through the Internet. Later, a teacher can analyze results and get conclusions about general level or of a particular student.

The project involves designing and implementing all the features available until now but integrated in a single modular application the three independent existing applications, and using an architecture that allows future extensions.

The design of InnovaCampus 2.0 has been done using more current technologies (Spring, JPA) and some features have been added.

Resum

InnovaCampus 2.0 és el redisseny de l'aplicació de programari lliure InnovaCampus utilitzant els frameworks Spring i JPA.

InnovaCampus és una aplicació que permet l'autoavaluació d'estudiants universitaris a través d'Internet. Posteriorment, un professor pot analitzar-ne els resultats i obtenir conclusions sobre el nivell general o d'un estudiant concret.

El projecte consisteix en dissenyar i implementar novament totes les funcionalitats existents fins al moment integrant en una única aplicació modular les tres aplicacions independents existents fins ara i usant una arquitectura que afavoreixi les ampliacions futures.

El disseny d'InnovaCampus 2.0 s'ha fet usant tecnologies més actuals (Spring, JPA) i s'hi han afegit algunes funcionalitats.

Índex

Índex de figures

Capítol 1

El projecte InnovaCampus

En l'actual era de la tecnologia en la que vivim, Internet és present en tots els àmbits, i per què no en l'estudi? Tot aprofitant l'atractiu del que disposa la xarxa, el projecte InnovaCampus planteja una eina de campus virtual en que cada estudiant pot autoavaluar-se de les seves assignatures, i el professorat pot obtenir una sèrie d'estadístiques per veure el rendiment i el nivell dels seus alumnes.

La finalitat d'InnovaCampus era molt bona, però havia estat desenvolupat com a tres aplicacions completament independents, fet que feia molt costoses les millores i el manteniment de l'aplicació web.

Aquest projecte sorgeix per la mancança de disposar d'una única aplicació molt modular, ben estructurada en una arquitectura de tres capes, ben documentada en javadoc, i preparada per a ser desenvolupada cooperativament com una aplicació de codi obert.

1.1 Història

InnovaCampus sorgí de la iniciativa del Dr. Joan Ribera, professor de la Facultat de Medicina de la Universitat de Lleida, que necessitava una aplicació amb els objectius de, per una banda oferir una eina d'autoavaluació pels estudiants, i per l'altra permetre al professorat el seguiment dels resultats dels seus alumnes.

Primerament es desenvolupà un prototipus bàsic inicial, aviat se'n va implementar un prototipus millorat afegint-hi funcionalitats (vegeu [1]), més endavant es millorà l'aplicació amb un mòdul que tractava la dependència entre preguntes i permetia afegir recursos a les preguntes (vegeu [2]), i un altre que canviava completament la concepció que hi havia fins al moment de consultar els resultats dels estudiants ja que incorporava gràfiques i estadístiques de tot tipus (vegeu [3]).

Aquests darrers mòduls també incorporaven una enginyeria del software basada en Struts, el patró DAO i el patró de disseny MVC.

La versió d'InnovaCampus que està corrent actualment, formada per les tres aplicacions recent esmentades, pot trobar-se a http://sedna.udl.cat:8080/InnovaCampus/.

1.2 Millores en l'enginyeria del software

Les millores en l'enginyeria del software d'aquest projecte passen per refer internament InnovaCampus per a que deixin d'ésser tres aplicacions independents i passin a ser una única aplicació implementada amb una arquitectura de tres capes, sobre el framework Spring i molt ben documentada per a futures ampliacions, particularment en comunitat.

Primerament és necessària una reenginyeria, des de la reestructuració, que passa per redissenyar la base de dades, replantejant el model de domini, i reimplementant tant les classes de serveis com la interfície.

Tot això s'ha de dur a terme en mode cooperatiu mitjançant un gestor de versions i es testejarà amb junit. En el servidor s'hi emmagatzemarà tota la documentació d'InnovaCampus (com desplegar l'aplicació, javadoc, model de domini, llicència, etc). I també s'hi aplicaran els patrons de disseny DAO i MVC.

1.3 Desenvolupament en comunitat

Eric Raymond [8] compara el disseny de software usualment propietari de gran envergadura amb la construcció de catedrals, és a dir, que hauria d'ésser cuidadosament el·laborat per genis o petites bandes de mags treballant tancats, sense alliberar versions beta abans d'hora.

L'estil de desenvolupament de software lliure, en canvi, es basa en alliberar ràpid i sovint, delegar tot el que es pugui i ésser obert fins al punt de la promiscuitat. Aquesta comunitat sembla més un bazar de Babel en ebullició, format per individus amb propòsits i enfocaments diferents, però d'on en sorgeix un sistema estable i coherent.

Per a començar la construcció d'un edifici comunal, el que s'ha de ser capaç de fer és presentar una promesa plausible. El programa no necessita ser particularment bo, pot tenir molts errors, pot estar incomplet i pobrement documentat, però en el que no pot fallar és en convéncer als codesenvolupadors potencials de que el programa pot evolucionar fins a alguna cosa elegant en el futur.

Com molt bé s'il·lusta en l'article "La Catedral y el Bazar", per a resoldre un problema interessant s'ha de començar per a trobar un problema que et resulti interessant.

Les infraestructures que requereix el desenvolupament en comunitat són: un repositori, en el nostre cas sedna http://sedna.udl.cat/svnInnovaSp, i un gestor de versions, d'entre els que hem escollit Subversion.

Tot el projecte InnovaCampus ha estat i continuarà implementat-se sota la Llicència GPL (vegeu B).

1.4 Objectius actuals

L'objectiu és fer un redisseny de l'aplicació InnovaCampus de tal manera que permeti continuar el desenvolupament en comunitat i modular com un projecte de codi obert [8], ja que les tres aplicacions existents fins al moment no eren homogènies a nivell de disseny intern. No es pretén aconseguir una aplicació completament funcional perquè això suposaria parlar d'un projecte d'una envergadura molt més àmplia que la que comprenen dos projectes de final de carrera, l'objectiu és obtenir un prototipus prometedor.

En particular, aquest redisseny pretén:

1. Millorar alguns aspectes d'enginyeria del software
    - Usant els frameworks Spring i JPA
    - Fent una aplicació extensiva dels patrons MVC i DAO
    - Proporcionant un desacoblament complet entre capes
2. Proporcionar una aplicació més modular que permeti una ampliació futura i en comunitat senzilla.
3. Proporcionar un prototipus prometedor per iniciar l'etapa de desenvolupament de l'aplicació en comunitat seguint un model similar al Bazar (vegeu [8]). Per això es necessita:
    - Documentació complerta per desplegar l'aplicació i Javadoc per totes les classes
    - Control de versions (SVN)
    - Repositori (Sedna)

1.5 Estructura de la memòria

La memòria d'aquest projecte està estrucurada en capítols i seccions. A continuació exposarem un breu resum de què conté cada capítol:

- Capítol 2: En aquest capítol s'estudien els requeriments a nivell general de l'aplicació, s'estudien quin són els casos d'ús, se n'obté el model de domini i es comenten una sèrie de requisits no funcionals.
- Capítol 3: En aquest apartat s'analitza com s'estructurarà la informació en la base de dades, com s'implementarà la persistència (amb el framework JPA) i com s'interaccionarà amb aquesta informació. S'explica com migrar de tecnologies i també s'enumeren les dificultats que ha comportat desenvolupar aquesta capa.
- Capítol 4: En el capítol de la capa de negoci ens hem centrat a explicar quines són les classes del domini necessàries per l'aplicació, quins són els subsistemes de la lògica de negoci per operar amb elles, com han sorgit i com s'han implementat. També s'esmenten quins han estat els problemes amb els que ens hem trobat per desenvolupar aquesta capa.

- Capítol 5: Aquesta part està més enfocada en la implementació de la interfície, el que visualitzarà l'usuari. Exposem què és el Model Vista Controlador, com s'utilitzen els controladors de Spring, la manera com estan dissenyades les vistes web (CSS,AJAX,Tiles), i com ho hem internacionalitzat.
- Capítol 6: En aquest capítol parlem de la seguretat a InnovaCampus, i de les classes per a generar tests i gràfiques.
- Capítol 7: Aquí exposem com hem testejat les classes de l'aplicació amb JUnit per comprovar-ne el correcte funcionament.
- Capítol 8: Explicació dels passos a seguir per part dels nous desenvolupadors de l'aplicació InnovaCampus.
- Capítol 9: En aquest apartat s'esmenten les conclusions a les que hem arribat durant el desenvolupament del projecte. També s'exposen aspectes a millorar i treball futur.
- Apèndix A: Breu descripció de les tecnologies utilitzades més rellevant, llenguatges, procediments i tècniques que s'han utilitzat durant la realització d'aquest projecte.
- Apèndix B: En aquest annex s'hi troba el text oficial de la Llicència GPL.
- Apèndix C: Mostra de la documentació en Javadoc d'InnovaCampus, des de captures de pantalla de l'índex de les classes fins als mètodes concrets.
- Apèndix D: Especificació de tots els casos d'ús dels subsistemes de la capa de negoci.

Capítol 2

Anàlisi de requeriments

L'anàlisi de requeriments és el primer pas en el desenvolupament d'una aplicació, ja que abans d'iniciar qualsevol mena de disseny és indispensable analitzar quines funcionalitats ha d'oferir. En el nostre cas es realitza mitjançant un anàlisi del context de l'aplicació actual, seguit d'un model de casos d'ús, que consisteix en identificar quines funcionalitats i restriccions ha de complir l'aplicació en cada cas, extraient un model de domini i finalment enumerant una sèrie de requeriments no formals.

2.1 Context general d'InnovaCampus

Els requeriments principals que pretén abordar aquest projecte són:

- Mantenir la funcionalitat general d'InnovaCampus
- Mantenir la correcta funcionalitat de les estadístiques que presenta
- Implementar l'aplicació web amb el framework Spring i accedint a la base de dades mitjançant JPA
- Proporcionar un desacoblament entre capes
- Dissenyar una jerarquia de tipus de test per a facilitar-ne l'ampliació en un futur

Funcionalitat general d'InnovaCampus:

- Canviar password d'usuari
- Recordar clau
- Modificar dades d'usuari
- Registrar-se
- Veure anuncis de l'administrador
- Veure anuncis del taulell d'anuncis d'una assignatura
- Donar d'alta un professor
- Donar de baixa un professor

- Donar d'alta un anunci
- Donar de baixa un anunci
- Donar d'alta una assignatura
- Donar de baixa una assignatura
- Modificar dades d'una assignatura
- Reiniciar una assignatura
- Donar d'alta un professor a una assignatura
- Donar de baixa un professor a una assignatura
- Donar d'alta un alumne a una assignatura
- Donar d'alta un llistat d'alumnes a una assignatura mitjançant un fitxer
- Donar de baixa un alumne d'una assignatura
- Donar d'alta un tema a una assignatura
- Donar de baixa un tema a una assignatura
- Modificar un tema d'una assignatura
- Donar d'alta un objectiu a un tema d'una assignatura
- Donar de baixa un objectiu d'un tema d'una assignatura
- Modificar un objectiu d'un tema d'una assignatura
- Donar d'alta una pregunta a un objectiu
- Donar de baixa una pregunta d'un objectiu
- Modificar una pregunta d'un objectiu
- Donar d'alta una resposta a una pregunta
- Donar de baixa una resposta d'una pregunta
- Donar d'alta un anunci al taulell d'anuncis d'una assignatura
- Donar de baixa un anunci
- Resoldre un test

Funcionalitat de les estadístiques:

- Calcular estadístiques per temes, objectius, preguntes i tests.
- Calcular estadístiques per a un alumne concret en temes, objectius, pregunts i tests.
- Estadístiques entre dues dates

- Generació dinàmica de gràfiques
- Exportar dades a fulls de càlcul i altres formats
- Estadístiques calculades (percentatges, desviacions típiques,etc.)
- Cercador d'alumnes

Requeriments de l'entorn:

- El sistema ha de poder operar sota qualsevol sistema operatiu.
- L'aplicació ha de detectar l'entrada errònia de dades.

Requeriments de la interfície:

- El sistema es comunicarà amb altres màquines mitjançant el protocol HTTP.
- Les dades estaran en HTML sempre que sigui possible.

Restriccions de disseny:

- Compliment dels estàndars XHTML, CSS i WAI-A.
- Ús d'AJAX per a simular una aplicació d'escriptori
- Ús de l'especificació JSP 2.0

La implementació de l'aplicació web amb Spring requereix especificar una sèrie de fitxers sobre el context de l'aplicació per a indicar-li què s'especifica en cada classe java. JPA implica afegir tags a les classes de domini que s'han d'emmagatzemar a memòria, a fi de poder treballar amb operacions de jpa, que són de més alt nivell i per tant més cómodes d'utilitzar que sql.

Finalment, el disseny d'una jerarquia de tipus de test comporta el plantejament de quines classes accediran a la base de dades, quines faran la selecció de les preguntes i quines seleccionaran les preguntes que han d'aparèixer al test, tot això reutilitzant la major part de codi possible i deixant-ho obert per a futures ampliacions.

2.2 Model de casos d'ús

Els casos d'ús s'utilitzen per a explicitar la seqüència de passos que han de seguir una sèrie d'actors per a realitzar una tasca. En InnovaCampus els actors són el professor, l'alumne, l'administrador i el propi sistema.

Seguidament mostrarem quins són els casos d'ús de les classes Service que pot realitzar cada actor i n'explicarem un detalladament. El detall de la resta es pot trobar a l'apèndix D.

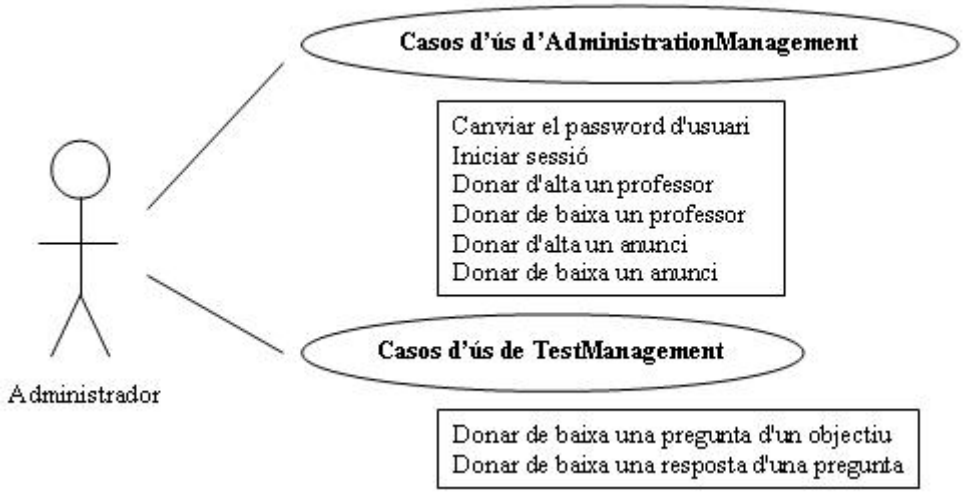

Figura 2.1: Casos d'ús de l'administrador

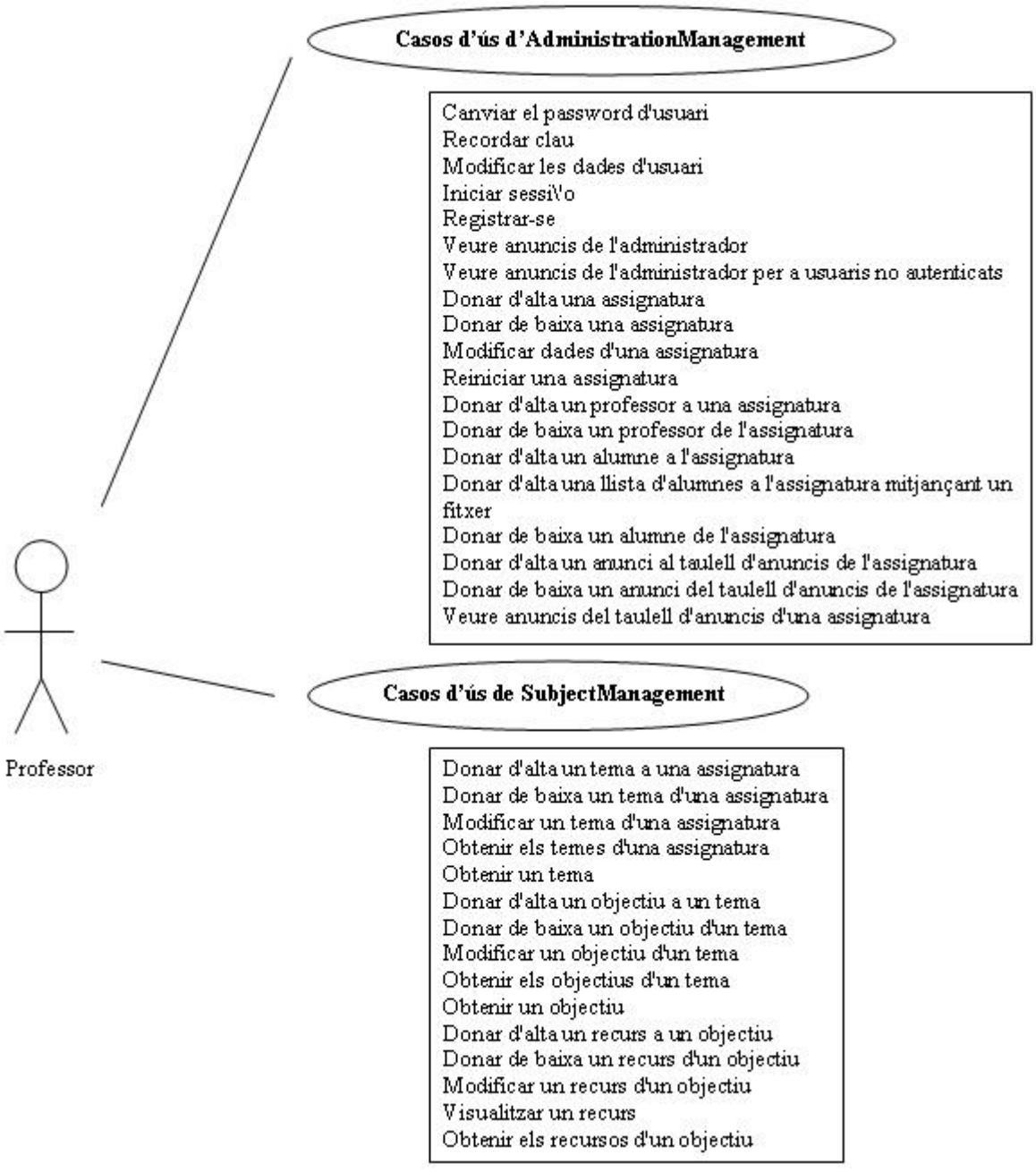

Figura 2.2: Casos d'ús del professor. Part I

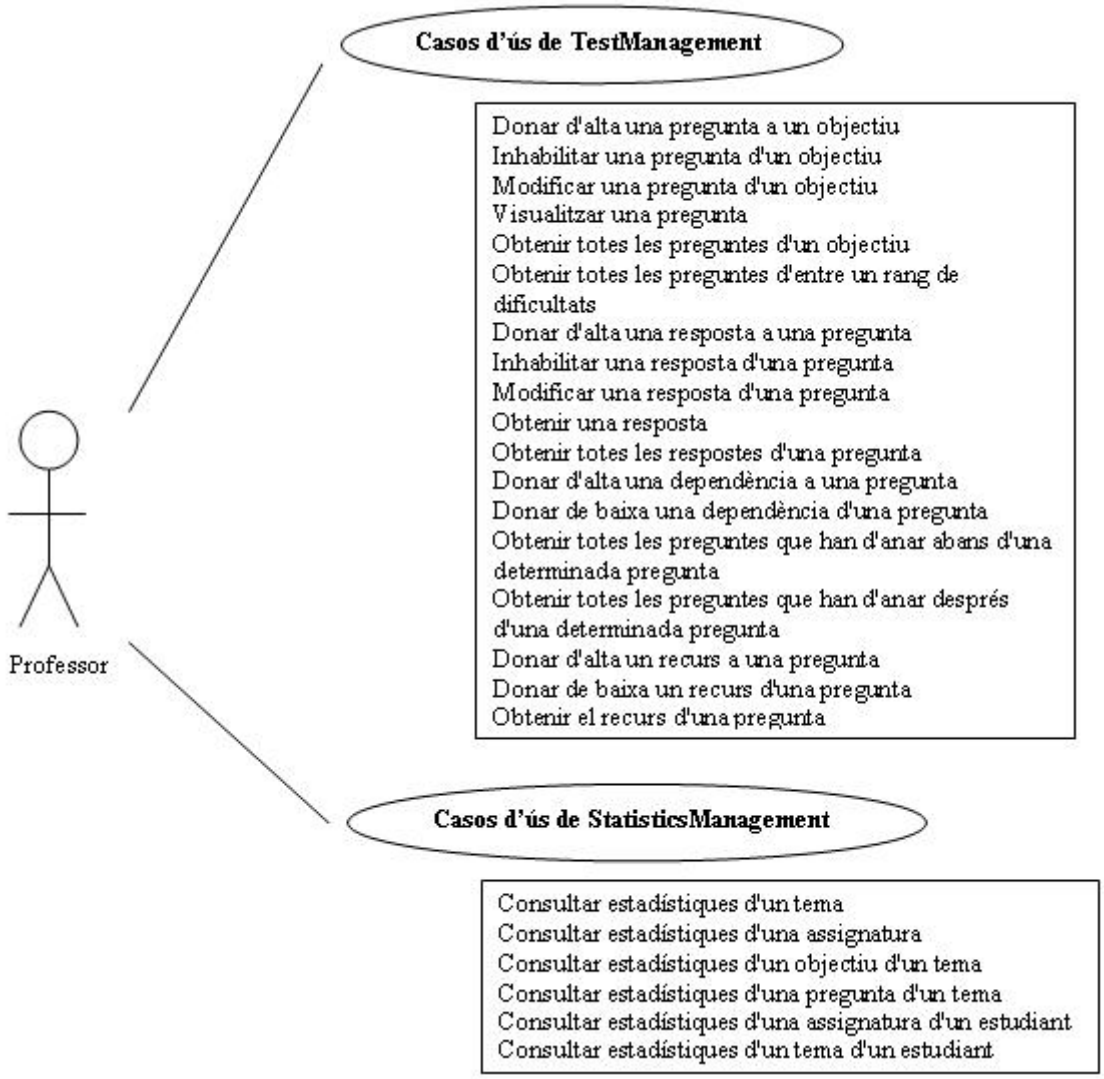

Figura 2.3: Casos d'ús del professor. Part II

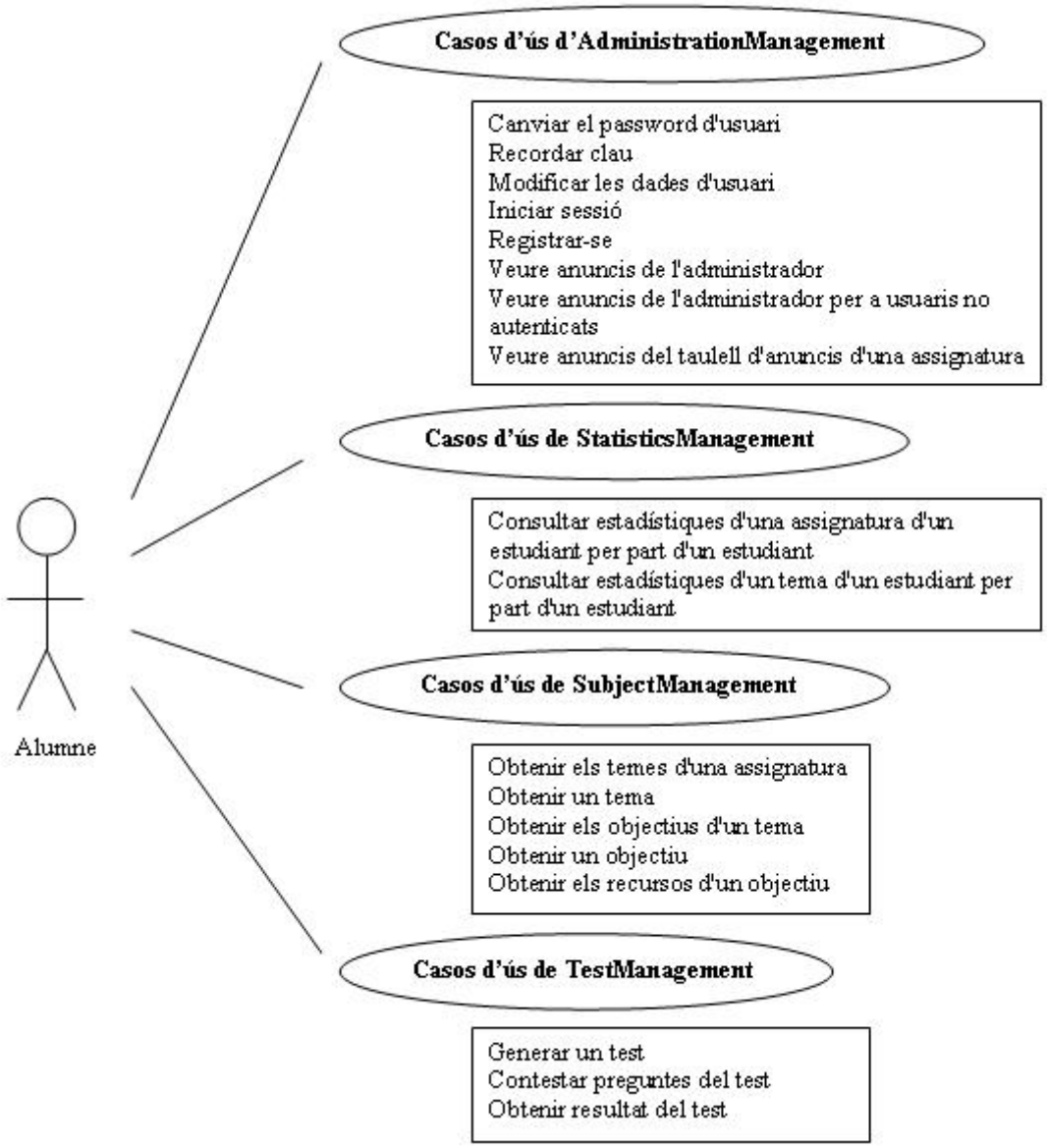

Figura 2.4: Casos d'ús de l'estudiant

Exemple d'un cas d'ús:

Cas d'ús:	**Registrar-se**
Descripció:	L'objectiu és que l'usuari es registri al sistema.
Actors:	Sistema i Professor o Sistema i Alumne
Precondicions:	El professor ha estat donat d'alta per l'administrador i l'alumne ha estat matriculat en alguna de les assignatures del sistema.
Postcondicions:	L'usuari ja està registrat al sistema i pot accedir al contigut de l'eina InnovaCampus.

Flux principal:	
Usuari	**Sistema**
1. L'usuari indica que vol registrar-se.	2. El sistema demana a l'usuari la seva identificació, correu electrònic i contrasenya.
3. L'usuari introdueix les dades.	4. El sistema registra a l'usuari i li envia un correu electrònic notificant-ho.

Casos d'error:
3. El correu electrònic no té un format vàlid. 3. La identificació de l'usuari no ha estat donada d'alta encara.

L'acció de registrar-se la poden dur a terme tant l'actor professor o com alumne. Primerament l'actor en qüestió ha d'introduir les seves dades: identificador i password, i si tot és correcte serà informat de que ha anat bé.

Els possibles errors que pot donar aquest cas d'ús són que el password o identificador no siguin correctes, error tan trivial que no es reflecteix en la descripció del cas d'ús, o que l'identificador no estigui donat d'alta. Aquest darrer cas es podria donar perquè és necessari haver donat d'alta en el sistema un professor per a que es pugui registrar. I en el cas de l'alumne, necessita haver estat matriculat en alguna de les assignatures.

2.3 Model del domini

El model de domini és una representació gràfica, extreta dels casos d'ús, de les entitats que formen el projecte i les relacions que s'estableixen entre elles.

Respecte al model de domini que hi havia anteriorment hem incorporat la jerarquia d'usuaris per a no tenir informació redundant i hem afegit diversos atributs a entitats ja existents. En l'entitat Answer hi hem afegit enabled a fi de que les respostes no s'eliminin del sistema fins que no ho decideixi l'Administrador, és a dir, que el Professor pot deshabilitar una resposta però no la pot esborrar. El mateix passa a Question. En aquesta última, a més, també hi hem afegit difficulty, timeAverage i answeredTimes per a poder disposar de la dificultat de la pregunta en cada moment i de dades interessants a l'hora de calcular la nova dificultat.

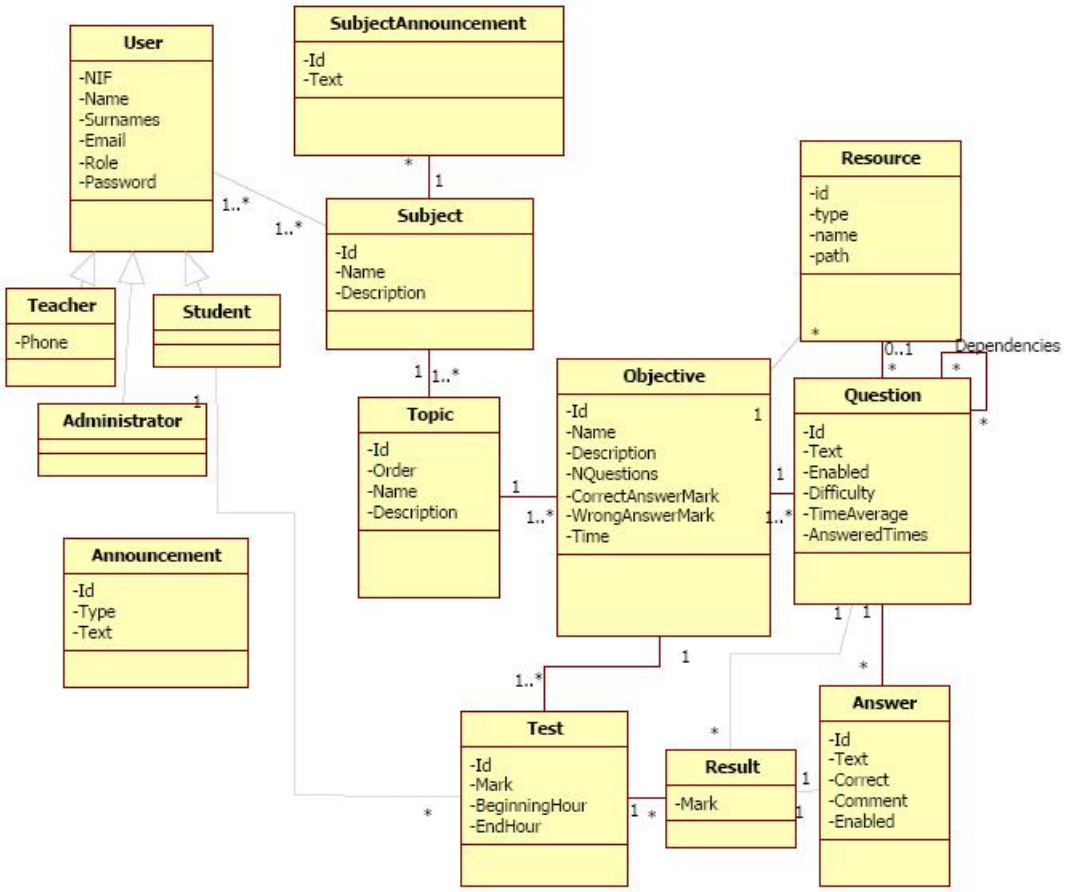

Figura 2.5: Diagrama del model de domini

2.3.1 Explicació del domini

A continuació descriurem breument la finalitat de cada entitat.

- User: Engloba les entitats Administrator, Teacher i Student, que comparteixen la majoria d'atributs.
- Administrator: És l'usuari amb més permisos i l'encarregat de supervisar el sistema.
- Teacher: És l'usuari que disposa de més funcionalitats. Cada professor és responsable d'una o vàries assignatures a les quals pot afegir preguntes, respostes, consultar-ne les estadístiques, matricular-hi alumnes i publicar-hi anuncis. S'identifica amb el NIF.
- Student: És l'usuari que s'encarrega de realitzar testos per autoavaluar-se. Cada estudiant pot estar matriculat a una o vàries assignatures, i pot consultar les seves estadístiques. S'identifica amb el NIF.
- Announcement: Aquesta entitat representa els anuncis globals del sistema, ja siguin per a usuaris no connectats, per a professors, per a estudiants o per a tothom, i és la utilitzada per l'administrador.
- Subject: Les assignatures estan dividides en temes i poden tenir anuncis associats.

- SubjectAnnouncement: Aquest tipus d'anuncis són els que publiquen els professors, i només fan referència a l'assignatura en qüestió.
- Topic: Els temes tenen un ordre i estan dividits en objectius.
- Objective: Aquestes entitats emmagatzemen la informació de quantes preguntes formaran un test, quina serà la puntuació aplicada en les respostes correctes i errònies, i el temps màxim per a resoldre un test. A més els objectius poden tenir recursos associats, preguntes i testos.
- Resource: Pot tractar-se d'imatges, vídeo, àudio o documents. D'aquests n'emmagatzemarem la ubicació i el títol.
- Test: L'entitat test només s'encarrega de les dades globals d'aquest, com són la nota, l'hora d'inici i l'hora de finalització. Cada test té varis resultats associats, un per a cada pregunta que formava el test.
- Result: Relaciona una pregunta, amb la resposta escollida, a un test associant-hi la nota obtinguda.
- Question: Les preguntes tenen un text, poden estar associades a algun dels recursos de l'objectiu en qüestió, poden dependre d'altres preguntes i tenen una sèrie de respostes.
- Answer: Cada resposta disposa del seu text, un possible comentari establert per part del professor i la indicació de si és una de les respostes correctes.

2.4 Requeriments no funcionals

En aquest apartat abordarem el tema dels requeriments no funcionals de l'aplicació. Tenint en compte que ha estat i serà utilitzada per estudiants de medicina, s'hauria de fer un disseny que tingués satisfés les seves necessitats en quant a usabilitat, i que corregís les mancances que ells hagin trobat.

També referent a la usabilitat fora bo que es replantegessin la carta de colors i el tipus de lletra. I sense oblidar que cada cop la universitat és mes global i l'intercanvi d'estudiants una grata realitat, la internacionalització de l'aplicació seria un pas endavant.

Capítol 3

Disseny: Arquitectura de l'aplicació i Capa persistència

3.1 Arquitectura de l'aplicació

El primer pas en el disseny d'una aplicació és decidir quina arquitectura s'utilitzarà. En el nostre cas hem optat per una arquitectura de tres capes: capa interfície, capa de negoci i capa persistència; el que pretén fonamentalment aquesta arquitectura és proporcionar-nos un desacoblament total entre cadascuna de les capes, fet que s'aconsegueix utilitzant interfícies per a que interactuin les capes entre elles.

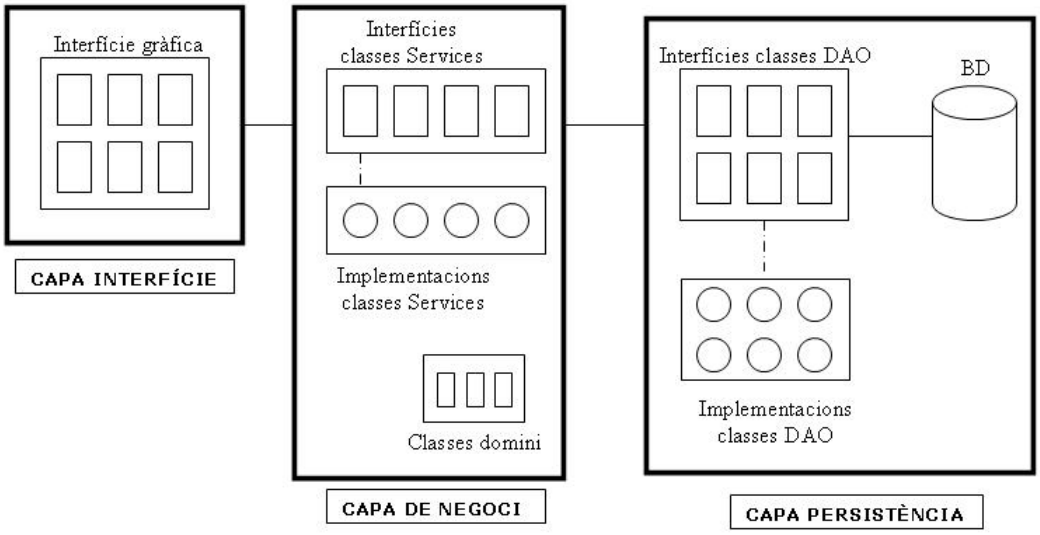

Figura 3.1: Arquitectura de tres capes

La capa de negocí és la que s'encarrega de tota la funcionalitat de l'aplicació. Està formada per les classes de domini i quatre subsistemes de la lògica de negoci, cadascun dels subsistemes disposa d'una interfície per a poder interactuar amb la capa interfície, i una implementació. Tal i com s'observa en la imatge, mitjançant una línia discontinua, la implementació pot ésser canviada sense que això afecti a la resta de les capes.

Tota la informació de l'aplicació s'emmagatzema en una base de dades relacional, ubicada en la capa de persistència, i per a ésser mappejada en el món dels objectes disposem d'interfícies DAO, amb les respectives implementacions. La capa interfície únicament conté les vistes de l'aplicació.

3.2 Introducció a la capa de persistència

La capa de persistència d'una aplicació és l'encarregada d'emmagatzemar les dades i facilitar l'accés a elles. Aquesta capa pot estar formada per un o més gestors de bases de dades que realitzen tot l'emmagatzemament de dades, reben les sol·licituds d'emmagatzemament i recupareció d'informació des de la capa de negoci.

En aquesta aplicació el gestor és el PostgreSQL, i les classes que interactuen direcament amb les dades són les classes DAO (Data Access Object) implementades en JPA (Java Persistence API).

3.3 Disseny de la base de dades

Els passos a seguir per a fer el disseny de la base de dades foren:

- Comparar l'estructura de la base de dades que estava corrent amb la que ens van facilitar els darrers desenvolupadors.
- Construir correctament la jerarquia User, Student, Teacher i Administrator.
- Afegir i modificar les taules i camps necessaris a partir de les modificacions realitzades en el model del domini.
- Traduir les taules i els camps a l'anglès.
- Crear camps enumerats

El disseny final de la base de dades està exposat a continuació, i es pot veure gràficament a la figura 3.1, i per facilitar-ne la comprensió es troben marcats en negreta els atributs referents a les claus primàries de cada entitat, i subratllades les claus foranes:

USERS (**nif**, name, surnames, password, role, email)
TEACHER (***nif***, phone)
STUDENT (***nif***)
ADMINISTRATOR (***nif***)
SUBJECT (**id**, name, description)
TOPIC (**id**, name, description, number, *subject_id*)
OBJECTIVE (**id**, name, description, nquestions, correct_answer_value, wrong_answer_value, max_time_per_test, *topic_id*)
RESOURCE (**id**, name, path, type, *objective_id*)
TEST (**id**, mark, beginning_time, end_time, *nif*, *objective_id*)
QUESTION (**id**, text, difficulty, answered_times, time_average, *objective_id*, *resource_id*, enabled)
ANSWER (**id**, text, comment, correct, enabled, *question_id*)

RESULT (***test_id***, *question_id*, *answer_id*, mark)
RESPONSIBLE (***subject_id***, ***nif***)
ENROLMENT (***subject_id***, ***nif***)
DEPENDENCIES (***question_id_a***, ***question_id_b***)
ANNOUNCEMENT (**id**, text, type)
SUBJECT_ANNOUNCEMENT (**id**, text, *subject_id*)
DIFFICULTY (***question_id***, ***nif***, *subject_id*, *topic_id*, *objective_id*, correct_answered_times, answered_times, student_difficulty))

Els canvis realitzats respecte al disseny anterior són:

- Estructurar la jerarquia d'usuaris
- Afegir els camps difficulty, answered_times i time_average a la taula question per agilitzar el càlcul de dificultats
- Afegir el camp enabled a la taula question i a l'answer a fi de que la informació persisteixi en la base de dades fins que l'administrador ho consideri oportú, principalment fins que es reinicialitzi l'assignatura, per no alterar les estadístiques
- Crera la taula difficulty per agilitzar l'obtenció de testos recomanats

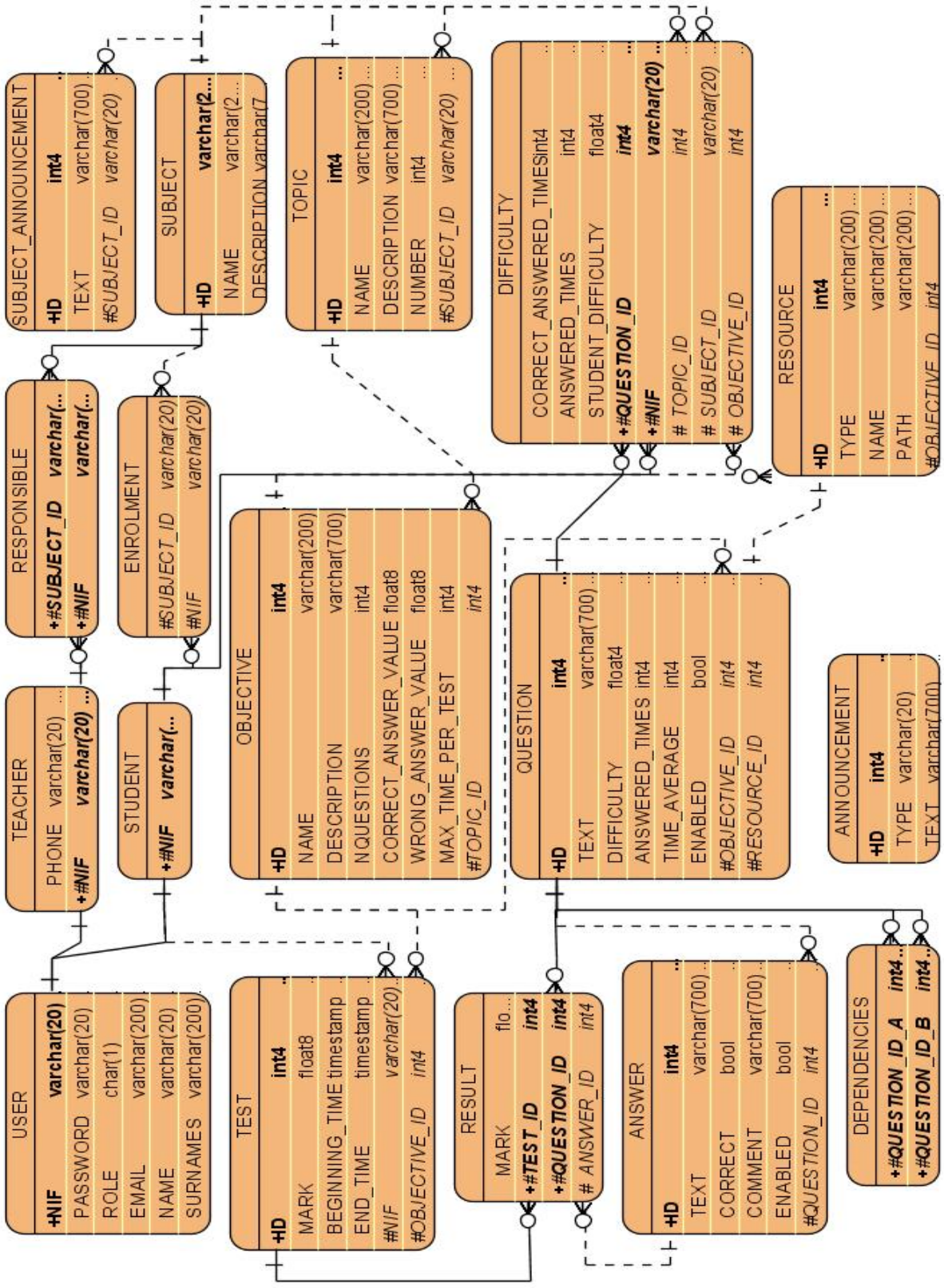

Figura 3.2: Disseny de la base dades d'InnovaCampus

3.4 ORM: JPA

Fins al moment, en les tres aplicacions que formaven InnovaCampus, no hi havia definida una bona capa de persistència. Nosaltres hem optat per fer un *mappeig* de les classes del domini mitjançant Object Relational Mapping(ORM) i implementant-ho utilitzant el patró de Data Access Object (DAO). En aquesta secció explicarem en que consisteix ORM i després exposarem el patró DAO.

ORM es basa en desenvolupar codi Java i mapejar-lo a la base de dades. Es poden utilitzar diferents tecnologies, com Hibernate o JPA. En el nostre cas hem fet servir JPA.

Per veure bé com es fa servir ORM i més concretament JPA explicarem el concepte d'entitats i com s'interrelacionen entre elles, com Spring i JPA interaccionen mitjançant EntityManager, i finalment el Java Persistence Query Language (JPQL) utilitzat en JPA.

3.4.1 Entitats i relacions

Cada interfície DAO disposa d'una implementació en JPA. Java Persistence API (JPA) està formada per:

- L'API de persistència.
- El Java Persistence Query Language (JPQL)
- Les metadades objecte/relació

En primer lloc cal implementar les entitats mitjançant classes lleugeres en Java, que acostumen a representar una taula en la base de dades, i cadascuna d'aquestes ha de contenir els atributs de la taula. Davant de cada atribut s'han d'inserir les metadades corresponents per indicar quin tipus d'atribut és i quines relacions té amb atributs d'altres taules. Això també pot indicar-se a través d'un arxiu XML. A InnovaCampus hem utilitzat les classes de domini com a entitats.

Un cop està tot enllaçat ja es pot començar a desenvolupar codi en JPQL o operacions definides per defecte en JPA.

Seguidament mostrarem i explicarem el codi desenvolupat per a la persistència d'estudiants.
User.java:

```
@Entity
@Table(name="USERS")
@Inheritance(strategy=InheritanceType.JOINED)
@DiscriminatorColumn(name="ROLE", discriminatorType=DiscriminatorType.STRING, length=1)

public class User implements Serializable {

@Id
private String NIF;
@Column(name="PASSWORD", nullable=false)
private String Password;
@Column(name="EMAIL", nullable=true)
```

```
private String Email;
@Column(name="NAME", nullable=true)
private String Name;
@Column(name="SURNAMES", nullable=false)
private String Surnames;
@Column(name="ROLE", nullable=false)
private char Role;

[...]
}
```

En primer lloc indiquem que es tracta d'una entitat mitjançant @Entity, seguidament la taula a la que fa referència amb @Table i el nom d'aquesta, especifiquem que forma part d'una jerarquia amb @Inheritance, i l'atribut que actua com a discriminador de la jerarquia amb @DiscriminatorColumn.

Finalment a cada camp de la classe hi col·loquem l'etiqueta @Column per relacionar-lo amb la columna de la taula de la base de dades. En el cas de la clau primària s'utilitza @Id, i no s'indica a quin atribut fa referència perquè utilitza el mateix nom (tenint en compte que Java és case sensitive).

Student.java:

```
@Entity
@Table(name="STUDENT")
@DiscriminatorValue(value="S")
public class Student extends User implements Serializable
{

@OneToMany(mappedBy="student",cascade=CascadeType.REMOVE)
private List<Test> tests = new LinkedList();
@ManyToMany
@JoinTable(name="ENROLMENT",
joinColumns = @JoinColumn(name="NIF",referencedColumnName="NIF"),
inverseJoinColumns = @JoinColumn(name="SUBJECT_ID",referencedColumnName="ID"))
private List<Subject> subjects = new LinkedList();

[...]
}
```

A banda del que hem explicat en User.java aquí podem observar l'etiqueta @DiscriminatorValue que indica que Student engloba els Users amb role='S'.

Algunes de les etiquetes més útils són les que representen les relacions, ja sigui d'u a molts (@OneToMany) o de molts a molts (@ManyToMany). En aquest exemple s'especifica que un estudiant pot tenir molts tests, i que en esborrar un estudiant s'eliminaran els seus tests en cascada. En la següent línia s'indica que un estudiant pot estar matriculat a vàries assignatures, i que una assignatura pot tenir varis estudiants.

3.4.2 EntityManager i Spring

EntityManager és la interfície que permet interactuar amb el context de persistència. Un context de persistència és un conjunt d'instàncies d'entitats on a cada entitat de persistència

li correspon una única instància. El EntityManager s'encarrega de controlar les instàncies de les entitats i els seus cicles de vida. La interfície d'EntityManager defineix els mètodes per interactuar amb el context de persistència, ja sigui per crear i eliminar entitats, buscar-ne o fer consultes sobre aquestes entitats.

```
<bean id="entityManagerFactory"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
<property name="dataSource" ref="dataSource" />

<property name="loadTimeWeaver">
<bean class="org.springframework.instrument.classloading.ReflectiveLoadTimeWeaver"/>
</property>

<property name="jpaVendorAdapter">
<bean class="org.springframework.orm.jpa.vendor.TopLinkJpaVendorAdapter">
<property name="showSql" value="false"/>
<property name="generateDdl" value="false"/>
<property name="databasePlatform" value=
"oracle.toplink.essentials.platform.database.PostgreSQLPlatform"/>
</bean>
</property>
</bean>
```

En aquest fragment de codi del fitxer applicationContext.xml s'hi indica el EntityManager, que s'utilitza TopLink i que gestor de base de dades és el PostgreSQL.

3.4.3 JPQL

Java Persistence Query Language (JPQL) és un llenguatge *case sensitive* de consulta orientat a objectes independent de plataformes que està definit com una part de l'especificació de JPA.

JPQL està fortament inspirat en SQL, la seva sintaxis així ho demostra, però opera amb entitats JPA enlloc de fer-ho amb les taules de la base de dades, i això permet escriure consultes portables que funcionen independentment de la base de dades subjacent i també polimorfisme. Suporta consultes del tipus SELECT, UPDATE i DELETE.

A continuació presentarem un exemple de sentència JPQL:

```
getJpaTemplate().find("select st from Student st, Subject sj where st.Name=?2 and
st.subjects=sj and sj.Id=?1",subjectId,name);
```

En aquest exemple s'obté l'estudiant amb nome *name* matriculat a l'assignatura *subjectId*. Per aconseguir-ho es fa un select però en el from s'indiquen les entitats, no les taules de la base de dades, i en el *where* es treballa amb els atributs de les entitats, no amb els camps de les taules com en SQL. Això proporciona independència entre la base de dades i la implementació de les classes que hi interaccionen, havent de modificar únicament, si varia alguna cosa en la base de dades, les etiquetes que es troben en l'entitat.

3.5 Patró DAO

El Patró DAO es basa en un raonament molt senzill però a la vegada molt útil. La idea és que la lògica de negoci sigui independent de la base de dades, és a dir, que no hi treballi directament sinò que ho faci amb una interfície que tingui els mètodes necessaris per a obtenir, modificar i eliminar la informació d'aquesta, amb la finalitat de que la interfície pugui disposar de vàries implementacions i que els canvis a la tecnologia de persistència usada i fins i tot, a modificacions lleus al disseny de la base de dades siguin transparents a l'aplicació.

InnovaCampus disposa d'una interfície DAO per a cada taula de la base de dades, i sent fidels al que acabem d'exposar, la capa de negoci accedeix a les dades únicament a través d'aquestes interfícies.

StudentDAO.java: StudentDAO és la interfície per a interaccionar amb les dades d'estudiants de la base de dades.

```
public interface StudentDAO {

  public void addStudent(Student s);

  public Student findStudentById(String studentId);

  public List findStudentByName(String subjectId, String name);

  public List findStudentsBySubject(String subjectId);

  public void updateStudent(Student s);

  public void delelteStudent(String studentId);
}
```

3.6 Com migrar de tecnologies

Si en algun moment donat es decideix migrar de tecnologies, ja sigui canviar de gestor de base de dades, de ORM o utilitzar una altra classe que implementi una interfície DAO, Spring ens ho posa molt fàcil.

Per exemple, per indicar que la implementació de la interfície AnnouncementDAO es troba a la classe JPAAnnouncementDAO només cal afegir les següents sentències add/spring/applicationContext.xml:

```
<bean id="jpaAnnouncementDao" class="org.innovacampus.dao.jpa.JPAAnnouncementDAO">
<property name="entityManagerFactory" ref="entityManagerFactory" />
</bean>

<bean id="administrationManagement"
class="org.innovacampus.services.impl.AdministrationManagementImpl">
[...]
<property name="announcementDAO" ref="jpaAnnouncementDao" />
```

```
[...]
</bean>
```

I si decidíssim utilitzar una implementació en Hibernate que es troba a HIBERNATEAnnouncementDAO doncs hauríem d'incloure la llibreria d'Hibernate al directori lib/libTomcat i actualitzar el fixter dd/spring/applicationContext.xml (vegeu 3.4.2) especificant la ruta de la nova classe i associant-li a la interfície announcementDAO com es mostra a continuació.

```
<bean id="hbAnnouncementDao" class="org.innovacampus.dao.hibernate.HIBERNATEAnnouncementDAO">
<property name="entityManagerFactory" ref="entityManagerFactory" />
</bean>

<bean id="administrationManagement"
class="org.innovacampus.services.impl.AdministrationManagementImpl">
[...]
<property name="announcementDAO" ref="hbAnnouncementDao" />
[...]
</bean>
```

Si el que pretenem és canviar el gestor de base de dades, per exemple per MySQL, hauríem d'especificar-ho al fitxer properties/jdbc.properties, indicant la url de MySQL, on es troba el driver, l'usuari i la contrasenya. També hauríem de referenciar la llibreria de MySQL i modificar el fitxer dd/spring/applicationContext.xml configurant el EntityManager per a que treballi amb el gestor de la base de dades MySQL.

```
<bean id="dataSource"
class="org.springframework.jdbc.datasource.DriverManagerDataSource">
[...]
</bean>

<bean id="entityManagerFactory"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
[...]
<bean class=
"org.springframework.orm.jpa.vendor.TopLinkJpaVendorAdapter">
[...]
<property name="databasePlatform" value=
"rutaDeLaPlataformaMySQL"/>
</bean>
[...]
```

Canviar el gestor de base de dades implica haver de canviar també la implementació de les classes DAO, i per tant en el fitxer dd/spring/applicationContext.xml ho hem d'indicar.

```
<bean id="msAnnouncementDao" class="org.innovacampus.dao.mysql.MYSQLAnnouncementDAO">
<property name="entityManagerFactory" ref="entityManagerFactory" />
</bean>
```

```
<bean id="administrationManagement"
class="org.innovacampus.services.impl.AdministrationManagementImpl">
[...]
<property name="announcementDAO" ref="msAnnouncementDao" />
[...]
</bean>
```

3.7 Transaccions

A l'hora d'implementar InnovaCampus hem tingut en compte les transaccions a nivell d'accés a la base de dades, mitjançant el getTransaction().begin i getTransaction().commit tal i com es mostra en l'exemple, però no les transaccions a nivell de negoci.

JPAStudentDAO.java:

JPAStudentDAO és una implementació en JPA de la interfície StudentDAO.

```
@Transactional
public class JPAStudentDAO extends JpaDaoSupport implements StudentDAO{

public void addStudent(Student s) {
getJpaTemplate().persist(s);
}

public void deleleteStudent(String studentId) {
Student s = findStudentById(studentId);
if(s!=null){
EntityManager em = getJpaTemplate().getEntityManagerFactory().createEntityManager();
em.getTransaction().begin();
em.createNativeQuery("DELETE FROM DIFFICULTY WHERE NIF='"+studentId+"'").executeUpdate();
em.getTransaction().commit();
getJpaTemplate().remove(s);
}
}

[...]

public List findStudentByName(String subjectId, String name) {
return getJpaTemplate().find("select st from Student st, Subject sj where st.Name=?2 and
st.subjects=sj and sj.Id=?1",subjectId,name);
}

[...]
}
```

En aquest fitxer hi apareixen operacions predefinides de JPA com getJpaTemplate().persist(s) que emmagatzema directament una entitat, o em.getTransaction().begin() i em.getTransaction().commit() que indiquen principi i fi d'una transacció.

En la darrera línia hi ha una consulta en JPQL, que com es pot comprovar, no és SQL.

3.8 Dificultats

Les dificultats amb les que ens vam trobar a nivell de la base de dades foren:

1. Vàrem haver de canviar el nom de taula USER per USERS i en la taula TOPIC el nom de l'atribut ORDER per NUMBER, ja que les anteriors corresponien a paraules restringides.

2. A nivell d'aplicar JPA el principal problema va estar trobar informació perquè és una tecnologia força nova. Després vàrem haver de substituir la llibreria InstrumentationLoadTimeWeaver per ReflectiveTimeWeaver degut a que dóna menys problemes per treballar amb transaccions. També ens va potar de cap el fet de que si declaràvem una relació ManyToMany continuava entenent-la com a unidireccional, i per arreglar-ho vàrem canviar el mappedBy per tot el codi en qüestió (JoinTable[...]). Aquest problema ens sorgí en les relacions:

   - questionA.Question - questionB.Question
   - subjects.Student - students.Subject
   - subjects.Teacher - teacher.Subjects

3. La darrera dificultat fou resoldre el mappeig de claus primàries compostes, que es donava en la taula RESULT amb TEST_ID i QUESTION_ID. La solució va estar crear l'atribut testId que no s'inclou en la base de dades (perquè es relaciona amb l'atribut test) i que no té cap relació. I amb QUESTION_ID vam adoptar el mateix mètode.

Capítol 4

Disseny: Capa de negoci

La capa de negoci és la que implementa la 'lògica de negoci' de l'aplicació. Aquesta 'lògica de negoci' correspon essencialment a les funcionalitats que l'aplicació ofereix als seus clients. Aquesta capa rep les sol·licituds de la interfície per a retornar-li posteriorment els resultats, i es comunica amb la capa de persistència per interactuar amb la informació emmagatzemada.

Tota aquesta lògica ha estat implementada en un conjunt de classes que les hem aplegat dins el paquet *services*, ja que englobarien tota la funcionalitat i serveis que dóna l'aplicació. Dins d'aquest paquet, les funcionalitats estan agrupades temàticament; d'aquesta manera ens han sortit 4 subsistemes, cadascú encarregat d'una tasca diferent: la part més administrativa de l'aplicació (*AdministrationManagement*), la part encarregada de les assignatures (*SubjectManagement*), la part encarregada dels tests (*TestManagement*) i la part encarregada de les estadístiques (*StatisticsManagement*).

Els subsistemes implementen totes les funcionalitats que ha de complir l'aplicació, controlant totes les excepcions que poden donar-se en cada mètode i utilitzant les interfícies de les classes DAO per obtenir, modificar i eliminar la informació de la base de dades. Per poder interactuar correctament amb les classes DAO i amb la interfície s'utilitzen les classes del domini. Aquestes classes representen els conceptes que apareixen al domini de l'aplicació.

Cada subsistema disposa d'una interfície per estructurar una arquitectura de tres capes correcta, ja que això permet que si en un moment donat es decideix canviar-ne la implementació, la capa d'interfície no se'n ressentiria perquè interactua amb la interfície de la capa de negoci.

Tot seguit veurem quines classes formen part del domini i de què s'encarrega cada subsistema.

4.1 Classes del domini

Com hem dit abans, les classes del domini representen els usuaris que interaccionen amb l'aplicació i els objectes amb els quals interaccionen.

Moltes vegades les classes del domini contenen informació que s'ha de guardar i per tant al mateix temps que són classes de domini també són entitats que han de persistir en una base de dades. Però no sempre és així, ja que podríem tenir objectes amb els quals interactua l'usuari però que no ens interessa guardar la seva informació, com per exemple quan es fa una compra online, l'objecte del carret de la compra és un objecte que mentre anem fent la compra l'usuari interacciona amb ell (afegint o traient productes) però que un cop l'usuari ha fet la comanda ja no el necessitem i l'objecte en sí del carret de la compra no cal guardar-ho en una base de dades. O, per exemple, objectes o coses que sempre seran de la mateixa forma, mai canviaran. Aquests objectes no cal guardar-los en una base de dades, perquè sempre sabrem com seran.

Llistat de classes que formen part del domini (en l'apartat 2.3.1 hi ha una breu descripció de cada classe):

- Administrator
- Announcement
- Answer
- Objective
- Question
- Resource
- Result
- Student
- Subject
- SubjectAnnouncement
- Teacher
- Test
- Topic
- User

4.2 Els subsistemes

A l'analitzar tota la funcionalitat que ha de tenir InnovaCampus, vam veure que hi havia molta cosa i vam decidir ordenar-ho per grups. Tenint en compte els objectius bàsics d'InnovaCampus i els seus mòduls vam decidir dividir-ho en quatre parts.

De la mateixa manera que a moltes aplicacions web, sempre hi ha una part més administrativa, que en el nostre cas s'encarregaria dels usuaris, anuncis i la creació d'assignatures, aquesta seria una part. Una altra part fa referència a un element important dins d'InnovaCampus, que són les assignatures. I les altres dues parts corresponen als dos mòduls que hi havia fins ara, el de la generació de tests i el de les mostrar les estàdistiques.

Tenint en compte tot el que s'ha dit, els subsistemes resultants són:

- **AdministrationManagement**: aquest subsistema s'encarrega de la part administrativa de l'aplicació.
  Gestiona els usuaris (tant professors com alumnes), donar-los d'alta, obtenir i modificar les seves dades i autenticar-los quan volen entrar a l'aplicació. Controla la creació d'anuncis, i obtenir i modificar la seva informació. Resol la creació d'assignatures, així com matricular alumnes i assignar pofessors a una assignatura.

- **SubjectManagement**: aquest subsistema es centra més en l'assignatura.
  S'encarrega d'afegir, modificar i esborrar temes i objectius. Una assignatura està dividida en temes i a la vegada els temes estan dividits en objectius. Aquest subsistema també s'encarrega de la gestió dels recursos (videos, imatges o documents). Un recurs s'associa a un objectiu en concret.

- **TestManagement**: aquest subsistema s'ocupa de la part dels tests.
  És el responsable de la gestió de preguntes i respostes. Les preguntes formen part d'un objectiu en concret i cada pregunta té múltiples respostes. També s'encarrega d'associar un recurs a una pregunta i de la generació de tests.

- **StatisticsManagement**: aquest subsistema s'encarrega de la part d'estadístiques, d'obtenir tota la informació necessària per a que el professor pugui veure els resultats dels tests de forma gràfica.
  obtenir el resultats d'un test, saber quants estudiants han realitzat tests d'un objectiu o tema, saber quants han aprovat o suspès, mirar els resultats d'un període de temps determinat.

Cada subsistema disposa d'una interfície i la seva corresponent implementació, d'aquesta manera es redueix l'acoblament entre les capes. Seguidament mostrarem un fragment de la interfície i la implementació del subsistema AdministrationManagement.

AdministrationManagement.java
És la interfície de la classe de serveis AdministrationManagement, on s'hi declaren els mètodes.

```
public interface AdministrationManagement {

public User getUser(String userId)
throws UserNotExistsException;

public Subject getSubject(String subjectId)
throws SubjectNotExistsException;

[...]

public void updateStudentData (String userId, String name, String surnames, String email)
throws UserNotExistsException;

[...]

public void signIn (String userId, String password)
throws UserNotExistsException, UserNotSignUpException, IncorrectPasswordException;
```

```
[...]

public void addResponsibleTeacherToSubject(String subjectId, String userId)
throws SubjectNotExistsException, UserNotExistsException, TeacherAlreadyResponsibleException;

[...]

}
```

AdministrationManagementImpl.java

És la classe que implementa la interfície AdministrationManagement.

```
public class AdministrationManagementImpl implements AdministrationManagement {

private UserDAO userDAO;
private TeacherDAO teacherDAO;
private StudentDAO studentDAO;
private AnnouncementDAO announcementDAO;
private SubjectAnnouncementDAO subjectAnnouncementDAO;
private SubjectDAO subjectDAO;

public User getUser(String userId)throws UserNotExistsException
{
User user = userDAO.findUserByNIF(userId);
if(user==null) throw new UserNotExistsException(userId,"AdministrationManagement.getUser");
return user;
}

public Subject getSubject(String subjectId) throws SubjectNotExistsException
{
Subject s = subjectDAO.findSubjectById(subjectId);
if(s==null) throw new SubjectNotExistsException(subjectId,"AdministrationManagement.getSubject");
return s;
}

[...]

public void updateStudentData (String userId, String name, String surnames, String email)
throws UserNotExistsException
{
User user = userDAO.findUserByNIF(userId);
if (user==null) throw new UserNotExistsException(userId,"AdministrationManagement.updateUserData");
else
{
Student student = studentDAO.findStudentById(userId);
student.setName(name);
student.setSurnames(surnames);
student.setEmail(email);
studentDAO.updateStudent(student);
}
}

[...]

public void signIn (String userId, String password)
throws UserNotExistsException, IncorrectPasswordException, UserNotSignUpException
{
User user = userDAO.findUserByNIF(userId);
```

```
if (user==null) throw new UserNotExistsException(userId,"AdministrationManagement.signIn");
else
{
if(user.getPassword()==null) throw new UserNotSignUpException(userId,
"AdministrationManagement.signUp");
if(!user.getPassword().equals(encryptPassword(password))) throw new
IncorrectPasswordException(userId,"AdministrationManagement.signUp");
}
}

[...]

public void addResponsibleTeacherToSubject(String subjectId, String userId)
throws SubjectNotExistsException, UserNotExistsException, TeacherAlreadyResponsibleException
{
Subject subject = subjectDAO.findSubjectById(subjectId);
if (subject==null)throw new SubjectNotExistsException(subjectId,"AdministrationManagement
.addResponsibleTeacherToSubject");
Teacher teacher  = teacherDAO.findTeacherById(userId);
if(teacher==null)throw new UserNotExistsException(userId,"AdministrationManagement
.addResponsibleTeacherToSubject");
List<Teacher> teachers = subjectDAO.findResponsibleTeachersBySubject(subjectId);
if(listContainsUser(teachers,teacher)) throw new TeacherAlreadyResponsibleException(userId,
subjectId, "AdministrationManagement.addResponsibleTeacherToSubject");
subjectDAO.addResponsibleTeachersToSubject(subjectId, userId);
}

[...]
}
```

Com es pot observar en el codi anterior les classes de serveis treballen amb les interfícies de les classes DAO i s'encarreguen de comprovar totes les possibles excepcions que puguin sorgir i informar-ne al client dels mètodes.

Com hem dit abans, treballar amb una interfície i la seva corresponent implementació redueix l'acoblament. En el fitxer *applicationContext.xml* se li indica quina és la implementació per a cada una de les interfícies dels subsistemes. Per exemple, la part del fitxer corresponent a la interfície *AdministrationManagement*:

```
<bean id="administrationManagement" class="org.innovacampus.services.impl.AdministrationManagementImpl">
    <property name="userDAO" ref="jpaUserDao" />
    <property name="teacherDAO" ref="jpaTeacherDao" />
    <property name="studentDAO" ref="jpaStudentDao" />
    <property name="announcementDAO" ref="jpaAnnouncementDao" />
    <property name="subjectAnnouncementDAO" ref="jpaSubjectAnnouncementDao" />
    <property name="subjectDAO" ref="jpaSubjectDao" />
  </bean>
```

Si es decidís canviar la implementació de la interfície i fer servir una altra classe, seria molt fàcil de fer. S'hauria de modificar el valor de l'atribut *class* i ficar el lloc on es troba el fitxer de la nova implementació. En principi les línies de propietats no caldria modificar-les perquè la nova implementació les podria necessitar. En cas contrari simplement s'esborren o es modifiquen les línies.

4.3 Dificultats

Els problemes amb els que ens hem trobat desenvolupant les classes de serveis no han estat amb la tecnologia utilitzada sinó amb l'essència dels mètodes en sí mateixos.

Per una banda el nivell de seguretat del xifrat dels *passwords*, ja que inicialment havíem optat per md5 però vàrem haver d'escollir-ne un de més robust (vegeu 6.1).

I la resta ja foren temes relacionats amb els testos. El fet de si era eficient emmagatzemar tests inacabats, al que finalment conclogueren que no, que era preferible crear els tests en software i, en finalitzar-los, el mateix estudiant podria decidir si s'ha d'emmagatzemar o no. Aquest tema s'ha d'acabar de perfilar en la capa d'interfície, però era important decidir-ho per implementar els mètodes de creació de tests.

Capítol 5

Disseny: Interfície

En aquest capítol veurem els aspectes de disseny de la interfície. Com el funcionament del patró Model Vista Controlador que hem fet servir per la gestió dels esdeveniments i selecció de les vistes a mostrar, la manera com estan dissenyades i com es generen les pàgines web, fent servir plantilles i AJAX, i l'ús de la internacionalització.

5.1 Model Vista Controlador (MVC)

El Model Vista Controlador (MVC) és un patró de disseny de software que separa les dades d'una aplicació, la interfície d'usuari, i la lògica de control en tres components diferents de forma que les modificacions al component de la vista poden ser realitzades amb un mínim impacte al component del model de dades. Això és útil ja que els models típicament tenen un cert grau d'estabilitat (depenent de l'estabilitat del domini del problema que està sent modelat), on el codi de la interfície d'usuari sigui més robust, degut a que el seu desenvolupador està menys proper a "trencar" el model mentre treballa de nou amb la vista.

El patró MVC és molt freqüent en el món de la web, on la vista és la pàgina que veu el client (HTML,JSP,XML) i el codi que proveeix de dades dinàmiques a la pàgina. En termes generals, construir una aplicació utilitzant una arquitectura MVC implica definir tres classes de mòduls.

- Model: Aquesta és la representació específica del domini de la informació sobre el qual funciona l'aplicació. La lògica de domini assegura la integritat de les dades i permet derivar-ne de noves. El model és una altra forma de cridar a la capa de domini.
- Vista: Aquesta presenta el model en un format adequat per a interactuar, usualment un element de la interfície d'usuari.
- Controlador: Aquest respon als esdeveniments, usualment accions de l'usuari i invoca canvis en el model i probablement a la vista.

Cadascun dels components de MVC té un propòsit específic. Per entendre més bé com funcionen examinarem el cicle de vida d'una petició típica en Spring MVC. Cada vegada que un usuari fa clic sobre un enllaç o envia un formulari en el seu navegador web, comença a treballar una petició. Una petició ho podem entendre com una informació que va d'un lloc a un altre.

La petició és com una persona molt ocupada. Des del moment en que abandona el navegador fins el moment en que retorna la resposta, farà varies parades, deixant cada vegada una mica d'informació i agafant-ne quelcom més. Les parades que realitza una petició són les que es presenten a la figura 5.1.

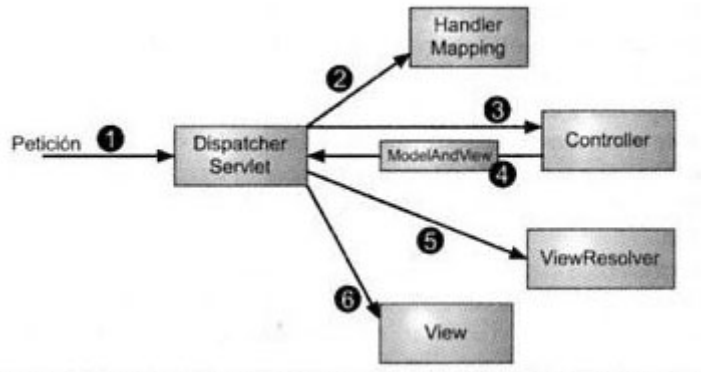

Figura 5.1: Cicle de vida d'una petició en Spring MVC.

Quan una petició surt del navegador, porta informació sobre el que està demanant l'usuari. La primera parada del viatge de la petició és el *DispatcherServlet* de Spring (1). Com la majoria dels marcs de treball MVC basats en Java, Spring MVC canalitza les peticions a través d'un únic servlet controlador frontal. Un controlador frontal és un patró comú de les aplicacions web on un únic servlet delega la responsabilitat d'una petició a altres components per a que realitzin el processament. En el cas de Spring MVC, *DispatcherServlet* és el controlador frontal.

La feina del *DispatcherServlet* és enviar la petició a un controlador de Spring MVC. Un controlador és un component de Sring que processa la petició. Però una aplicació típica pot tenir molts controladors i *DispatcherServlet* necessita ajuda per decidir a quin controlador enviar la petició. Així doncs, *DispatcherServlet* consulta un o més manipuladors de mappeig (2) per descobrir quina serà la següent parada de la petició. El manipulador de mappeig fa servir la URL transportada per la petició per prendre la seva decisió

Un cop s'ha escollit el controlador adequat, *DispatcherServlet* envia la petició al controlador escollit. En el controlador (3), la petició deixarà la informació enviada per l'usuari i esperarà pacientment a que el controlador processi la informació. (En el següent apartat veurem amb més detall el funcionament d'un controlador en Spring).

El controlador empaquetarà la informació que ha de retornar a l'usuari (*model*) i el nom d'una visualització (una pàgina web JSP) en un object *ModelAndView*. Llavors retornarà la petició juntament amb el seu nou paquet *ModelAndView* (4) al *DispatcherServlet*. Com indica el seu nom, l'objecte *ModelAndView* conté la informació que s'ha de mostrar a l'usuari i el nom de en quina *visualització* es mostraran els resultats.

Per a que el controlador no estigui acoblat a una visualització en particular, *ModelAndView* no porta una referència al JSP real. Porta un nom lògic que serà utilitzat

per buscar la visualització real que produirà el HTML resultant. Una vegada s'ha entregat el *ModelAndView* al *DispatcherServlet*, el *DispatcherServlet* li demanarà al resolutor de visualitzacions que l'ajudi a trobar el JSP (5).

Ara que el *DispatcherServlet* sap quina visualització donarà els resultats, la feina de la petició quasi ha acabat. La seva última parada és la implementació de la vista (probablement un JSP) en que entrega la informació que és per l'usuari. Amb aquesta informació entregada a la vista (6), acaba el viatge de la petició. La vista utilitzarà la informació que li ha donat la petició per mostrar una pàgina que serà retornada al navegador.

5.2 Controladors en Spring

Com hem vist abans, les principals funcions del controlador són: interceptar esdeveniments, fer operacions amb el model i establir quina ha de ser la següent vista. El que diferencia un controlador Spring MVC d'un servlet o una *Action* Struts és que està configurat com un altre JavaBean en el context de l'aplicació de Spring. Això vol dir que es pot aprofitar completament la injecció de dependència de Spring en una classe controlador de la mateixa manera que amb qualsevol altre bean.

La segona parada dins del cicle de vida d'una petició és el manipulador de mappeig. El *DispatcherServlet* consulta un o més manipuladors de mappeig per decidir a quin controlador li ha de passar la petició. El manipulador de mappeig consulta la URL que transporta la petició i així sap quin és el controlador corresponent. En el següent codi es mostra un exemple de les possibles URLs i el seu controlador associat:

```
<bean id="urlMapping"
  class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
  <property name="urlMap">
    <map>
      <entry key="login.htm"><ref bean="login"/></entry>
      [...]
      <entry key="teacher/home.htm"><ref bean="teacherHome"/></entry>
      <entry key="teacher/statisticsByTopic.htm"><ref bean="statisticsByTopic"/></entry>
      <entry key="teacher/showTest.htm"><ref bean="showTest"/></entry>
      [...]
    </map>
  </property>
  [...]
</bean>

<!-- Controladors -->
<bean id="login" class="org.innovacampus.mvc.controller.LoginController">
  <property name="administrationManagement" ref="administrationManagement"/>
</bean>
<bean id="teacherHome" class="org.innovacampus.mvc.controller.teacher.TeacherHomeController">
  <property name="administrationManagement" ref="administrationManagement"/>
</bean>
<bean id="statisticsByTopic" class="org.innovacampus.mvc.controller.teacher.StatisticsByTopicController">
  <property name="administrationManagement" ref="administrationManagement"/>
  <property name="subjectManagement" ref="subjectManagement"/>
  <property name="statisticsManagement" ref="statisticsManagement"/>
```

```
</bean>
<bean id="showTest" class="org.innovacampus.mvc.controller.teacher.ShowTestController">
  <property name="testManagement" ref="testManagement"/>
</bean>
```

Un cop el *DispatcherServlet* sap quin és el controlador, li envia la petició. Seguidament veurem el codi del controlador *TeacherHomeController*, que s'encarrega de mostrar la pàgina d'inici quan es connecta un profesor

```
public class TeacherHomeController extends AbstractController{

  private AdministrationManagement administrationManagement;

  public TeacherHomeController(){}

  protected ModelAndView handleRequestInternal(HttpServletRequest request,
    HttpServletResponse response) throws Exception {

    [...]

    String msg = new RequestContext(request).getMessage("inici");

    Teacher teacher = (Teacher) administrationManagement.getUser(id);

    List<Subject> subjects = teacher.getsubjects();

    ModelAndView mav = new ModelAndView("teacherHome");

    mav.addObject("listSubjects", subjects);

    mav.addObject("inici", msg);

    [...]

    return mav;
  }

  [...]

}
```

En aquest exemple es pot veure com retorna un objecte *ModelAndView* indicant-li que el nom de la vista és *teacherHome*. A més a més, com a informació se li passa la llista d'assignatures de les que és responsable aquest professor i un missatge de text. En aquest cas, el controlador té un atribut del tipus *AdministrationManagement* que no s'inicialitza en cap moment. Això és degut a que serà el propi contenidor de Spring qui s'encarregarà de crear i passar-li la instància al controlador.

5.3 Estructura de les pàgines

A l'hora de dissenyar InnovaCampus no s'ha seguit estrictament una metodologia de disseny centrada en l'usuari [4], però sí que s'ha considerat de forma important l'usuari i s'han tingut en compte diferents aspectes per tal de millorar l'usabilitat.

Aquesta és una aplicació que ja porta uns quants anys en funcionament, i l'experiència tant del profesor com de l'alumne s'han tingut molt en compte a l'hora de redissenyar l'aplicació i més concretament la part de la interfície. També hem comptat amb l'ajuda d'un dissenyador web que ens ha aconsellat sobre diversos aspectes de disseny de les pàgines web, com quins colors utilitzar, com posicionar la informació dins de la pàgina i quina mena de fonts són millors.

Tenint en compte com havia de ser la interfície, hem utilitzat diferents tecnologies per definir l'estructura de les pàgines web. Hem fet servir fulles d'estil en cascada (CSS) [10] per a la presentació de les pàgines web, d'aquesta manera se separa el contingut de la pàgina del propi disseny i l'estructura. Per la generació de les pàgines hem fet servir plantilles (Tiles[28]), així es pot fer servir per pàgines diferents la part de codi que és comuna. I per a una bona visualització de la informació en algunes pàgines hem fet servir AJAX. Ara explicarem més en detall el funcionament de les plantilles i d'AJAX.

5.3.1 Tiles

Jakarta Tiles és un marc de treball per dissenyar parts d'una pàgina en una plantilla. Tot seguit mostrem una part de la plantilla que fem servir en l'aplicació InnovaCampus:

```
<html>
[...]
  <body>
    <div id="web-frame">
      <!--capcalera-->
      <tiles:insertAttribute  name="header"/>
      <!-- Menu principal -->
      <tiles:insertAttribute name="main-menu"/>
      <!--Contingut-->
      <div id="content-frame">
        [...]
      </div>
      <!--fi content frame-->
      <hr />
      <!--footer-->
      <div id="footer">
        <tiles:insertAttribute name="footer"/>
      </div>
      <!--fi footer-->
    </div>
  </body>
</html>
```

S'utilitza l'etiqueta JSP Tiles *<tiles:insert>* per incloure contingut en la plantilla. Els detalls d'on s'origina el contingut inclòs s'especifiquen en el fitxer de configuració de Tiles, que és un fitxer XML que descriu com omplir la plantilla. El següent fragment mostra una part de la plantilla principal (anomenada *template*), omplint cadascun dels components:

```
<definition name="template" template="/WEB-INF/jsp/viewTemplate.jsp">
  [...]
```

```
    <put-attribute name="header" value="/WEB-INF/jsp/header.jsp"/>
    <put-attribute name="main-menu" value="/WEB-INF/jsp/menu.jsp"/>
    [...]
    <put-attribute name="footer" value="/WEB-INF/jsp/footer.jsp"/>
  </definition>
```

Aquí els components *header*, *main-menu* i *footer* prendran la ruta a fitxers JSP que defineixen com han de ser l'encapçalament, el menú principal i el peu. Quan Tiles construeixi la pàgina, substituirà les etiquetes *<tiles:insert>* anomenades *header*, *main-menu* i *footer* amb la informació de sortida de *header.jsp*, *menu.jsp* i *footer.jsp*, respectivament.

A partir de la plantilla *template* es poden anar originant diferents pàgines JSP segons com anem modificant les rutes que es defineixen als components. En aquest exemple, especifiquem les rutes que tindrà la pàgina d'inici d'un professor.

```
  <definition name="teacherHome" extends="template">
    [...]
    <put-attribute name="main-menu" value="/WEB-INF/jsp/teacher/main-menu.jsp"/>
    [...]
  </definition>
```

Estendre *template* garantitza que la pàgina heredarà totes les definicions dels seus components. No obstant, en aquest cas volem que el menú principal sigui un menú específic per als professors i per tant es defineix que la ruta per al component *main-menu* sigui *teacher/main-menu.jsp*. D'aquesta manera definim un pàgina nova amb els valors de la plantilla però amb el valor del menú principal diferent.

Per integrar Tiles a Spring MVC se li ha de dir a Spring que obri el fitxer(s) de configuració de Tiles. Spring disposa d'un bean que obre aquets fitxers i els fica a dispsició per mostrar vistes Tiles. Aquest bean s'anoma *TilesConfigurer* i es en la propietat *definitions* on es defineix la ruta al fitxer de configuració de Tiles, tal com es mostra a continuació:

```
<bean id="tilesConfigurer"
  class="org.springframework.web.servlet.view.tiles2.TilesConfigurer">
  <property name="definitions" value="/WEB-INF/InnovaCampusSp-tiles.xml"/>
</bean>
```

Per més informació sobre Tiles [28].

5.3.2 AJAX

L'AJAX (Asynchronous JavaSript and XML) no és pas una nova tecnologia, sinó tan sols un mètode d'aproximació per la interacció a la Web millorant l'experiència dels usuaris [6]. La principal idea d'aquesta metodologia és transmetre petits fragments d'informació entre el client i el servidor, afavorint així una resposta ràpida.

Per possibilitar això, JavaScript passa a gestionar la comunicació i a fer les peticions al servidor, el qual en comptes de retornar una resposta en HTML, ho fa en XML. Llavors, els mètodes JavaScript del client interpreten les dades i fan les modificacions adients al document que està visualitzant l'usuari.

En l'aplicació d'InnovaCampus hem fet servir AJAX en funcionalitats com navegació per pestanyes i càrrega d'altres pàgines en porcions de la vista actual. Això no ho hem hagut d'implementar nosaltres, sinó que hem fet ús del projecte AJAX Taglib.

El projecte AJAX Tag Library proporciona un conjunt d'etiquetes personalitzades que faciliten l'ús d'AJAX. Les etiquetes que es poden trobar en InnovaCampus són les següents:

- **htmlContent**: serveix per omplir una regió d'una pàgina amb codi HTML d'una altra obtingut amb AJAX. Exemple:

```
<div id="targetDiv"></div>
<div id="htmlContentForm">
  <select id="select" name="select">
    <option value="1">News</option>
    <option value="1">Sports</option>
    <option value="1">Weather</option>
  </select>
</div>
<ajax:htmlContent
  baseUrl="MyAction.htm"
  source="link"
  target="targetDiv"
  parameters="content={select}" />
```

  En aquest exemple, quan s'executi AJAX, s'omplirà la regió de la pàgina amb identificador *targetDiv* amb el contigut de la pàgina que correspon a *MyAction.htm*.

- **tabPanel:** ofereix una porció de pàgina amb diferents pestanyes. Quan l'usuari en selecciona una, es carrega el contingut corresponent. Exemple:

```
<ajax:tabPanel
  panelStyleId="tabPanel"
  contentStyleId="tabContent"
  currentStyleId="ajaxCurrentTab">
  <ajax:tab caption="First Tab"
    baseUrl="MyAction.htm"
    parameters="tab=1"/>
    defaultTab="true"/>
  <ajax:tab caption="Second Tab"
    baseUrl="MyAction.htm"
    parameters="tab=2"/>
  <ajax:tab caption="Third Tab"
    baseUrl="MyAction.htm"
    parameters="tab=3"/>
</ajax:tabPanel>
```

  En la pàgina hi haurà tres pestanyes i per defecte estarà activada la primera. En aquest exemple, les tres penstanyes tenen la mateixa url, però amb diferents valors per l'atribut tab. Així segons la pestanya que es premi es mostrarà un contingut o un altre.

La descripció de les etiquetes i els exemples estan basats en la documentació del projecte AJAX Taglib. Per més informació [7].

5.4 Internacionalització (i18n)

La internacionalització és el procés de dissenyar software de forma que pot adaptar-se a diferents idiomes i regions sense la necessitat de fer canvis en el codi [11].

Per solucionar el problema de la internalització hem decidit fer servir missatges externs. Com que és molt fàcil que apareixi el mateix text en vàries pàgines web, existeix la possibilitat de que cada ocurrència sigui inconsistent amb la resta, és a dir, que s'usin noms diferents en diferents llocs per referir-se al mateix concepte. També podria passar que es decidís canviar el text, llavors el canvi afectaria a cada pàgina en la que aparegués el text en concret.

Per tractar aquest problemes, Spring ofereix l'etiqueta <*spring:message*>. Com es mostra en la figura de baix, <*spring:message*> mostra un missatge des d'un fitxer de propietats de missatge extern a la informació de sortida. Fent servir <*spring:message*> es pot referenciar qualsevol missatge extern en la pàgina i mostrar el missatge actual quan es mostri la vista.

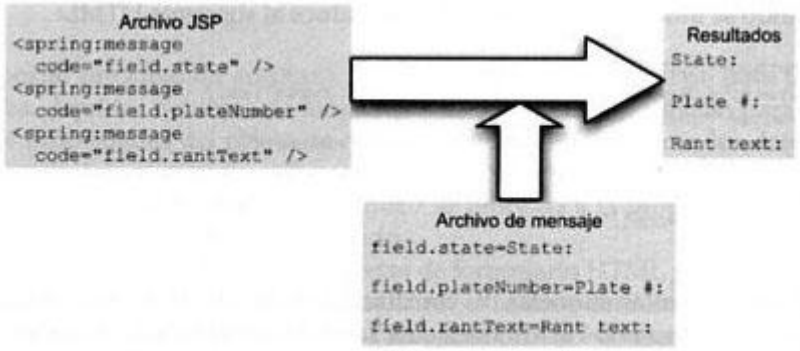

Figura 5.2: Funcionamet de l'etiqueta <*spring:message*>.

Per defecte, els missatges externs s'agafaran d'un fitxer anomenat *messages.properties*. Per cada un dels idiomes que es vulgui, es crea un fitxer de propietats indicant al nom l'idioma que volem: *messages_id.properties*, on *id* fa referència a l'idioma desitjat. Per exemple, si vulguéssim un fitxer de propietats en anglès, el nom del fitxer seria: *messages_en.properties*.

L'idioma en que es mostren les pàgines, per defecte seria en el que està configurat el nostre navegador. Per permetre que l'usuari pugui canviar l'idioma fent clic, el que es fa és que al fer clic s'envia una petició demanant que es canviï l'idioma mitjançant l'atribut *lang*. Per a que passi això s'ha de configurar Spring indicant-li que es farà servir internacionalització i indicant-li quin és el nom de l'atribut.

```
<bean id="localeChangeInterceptor"
  class="org.springframework.web.servlet.i18n.LocaleChangeInterceptor">
  <property name="paramName" value="lang"/>
</bean>

<bean id="localeResolver" class="org.springframework.web.servlet.i18n.SessionLocaleResolver"/>
```

De moment InnovaCampus està disponible en català, castellà i anglès. Però com hem dit anteriorment, afegir més idiomes a l'aplicació és senzill, simplement s'ha de crear un nou fitxer de propietats amb l'idioma desitjat i afegir l'opció a la vista perquè l'usuari el pugui escollir.

Capítol 6

Disseny: Altres aspectes

En aquest capítol veurem altres aspectes del disseny molt importants pel desenvolupament d'InnovaCampus.

La seguretat és un dels temes que hem tingut en compte en el redisseny de l'aplicació, hem considerat important xifrar les contrasenyes que es guarden a la base de dades, perquè fins ara es guardaven en text pla; i també hem fet servir filtres per poder tenir un control d'accés a les diferents pàgines web.

Un altre aspecte important és la generació de test, explicarem una mica en detall com està dissenyat i com funciona, i com es generen els tests per dificultat.

En el mòdul d'estadístiques s'utilitzen gràfiques per visualitzar més bé els resultats. En aquest capítol explicarem com es va fer el disseny de la generació de gràfiques i els canvis que hem hagut de fer per poder-ho adaptar a Spring.

6.1 Seguretat

Quan parlem de seguretat en aquesta aplicació fem referència a dos aspectes:

- Guardar les contrasenyes dels usurais xifrades.
- Tenir un control d'accés a les diferents pàgines web d'acord al tipus d'usuari.

Contrasenyes xifrades

Les contrassenyes dels usuaris es guarden xifrades a la base de dades. Inicialment vam establir un xifrat fent servir MD5 [21], però tenint en compte els resultats obtinguts per Xiaoyun Wang sobre la seva possible vulnerabilitat [29] vam decidir canviar el tipus de xifrat SHA-256 [22].

Control d'accés

Per comprovar que l'usuari i la contrassenya de l'usuari que s'autentica siguin correctes i per poder garantir que cada tipus d'usuari pugui accedir únicament a les pàgines que li corresponen, hem fet servir Spring Security (vegeu A.3).

Spring Security proporciona diferents mètodes per poder autenticar un usuari, però per poder fer servir els mètodes que nosaltres vam implementar hem hagut de reescriure alguns mètodes que proporcionava Spring Security per a que invoquessin als nostres mètodes d'autenticació i indicar-li a Spring Security que fes servir aquets mètodes.

Classe que s'encarrega d'invocar als nostres mètodes d'autenticació:

```
public class InnovaSecurity implements AuthenticationProvider {

  private AdministrationManagement administrationManagement;

  public Authentication authenticate(Authentication user) throws AuthenticationException {

    [...]

    try {
      administrationManagement.signIn(id,pass);
      User usuari = administrationManagement.getUser(id);
      role=role+usuari.getRole();
      return new UsernamePasswordAuthenticationToken(
        id,"",
        new GrantedAuthority[] { new GrantedAuthorityImpl(role) });
    }
    [...]
  }

  public void setAdministrationManagement(AdministrationManagement am){
    this.administrationManagement = am;
   }

}
```

Per indicar-li a Spring Security que faci servir la nostra classe, s'ha d'afegir a la llista de *providers* del controlador d'autenticació:

```
<beans:bean id="authenticationManager"
  class="org.springframework.security.providers.ProviderManager">
  <beans:property name="providers">
    <beans:list>
      <beans:ref local="myAuthenticationProvider" />
    </beans:list>
  </beans:property>
  [...]
</beans:bean>

<beans:bean id="myAuthenticationProvider" class="org.innovacampus.utils.InnovaSecurity">
  <beans:property name="administrationManagement" ref="administrationManagement" />
  [...]
</beans:bean>
```

Per garantir que cada tipus d'usuari pugui accedir únicament a les seves pàgines, Spring Security proporciona uns filtres per facilitar aquest control d'accés. Mostrarem un exemple de com s'utilitzen:

```
<intercept-url pattern="/home.htm" filters="none" />
<intercept-url pattern="/studentHome.htm" access="student" />
<intercept-url pattern="/**" access="teacher" />
```

Aquest exemple permet l'accés a tothom (sense filtres) per a la pàgina *home.htm*. En canvi, la pàgina *studentHome.htm* només és accecible per aquells usuaris que tinguin el rol *student*. Totes les altres pàgines són accecibles únicament per usuaris que disposin del rol *teacher*.

6.2 Generació dels testos

La generació de testos està dissenyada de forma que sigui fàcil integrar més tipus de tests a l'aplicació.

En general, quan es genera un test el que es fa és escollir les preguntes que sortiran al test d'un conjunt de preguntes. En el nostre cas, hem volgut separar aquesta idea en dues. Per una banda tenim el conjunt de preguntes (exemple: totes les preguntes d'un tema) i per altra banda hi ha la selecció d'aquelles que formaran part del test. Tenint en compte això i fent servir com a base el patró de *Dependency Injection* de Spring hem creat una classe *TestGenerator* que crida a funcions de dues interfícies: *ListOfQuestions* i *SelectionOfQuestions*.

- TestGenerator: aconsegueix una llista de preguntes a partir d'una classe del tipus *ListOfQuestions*. Aquesta llista la passa a una classe del tipus *SelectionOfQuestions* i obté les preguntes que sortiran al test.
- ListOfQuestions: és un conjunt de classes que implementen l'obtenció de la llista de preguntes d'una assignatura, d'un tema o d'un objectiu i si es volen totes les preguntes o tenint en compte el grau de dificultat.
- SelectionOfQuestions: és un conjunt de classes que, donada una llista de preguntes, obtenen les preguntes que sortiran al test, ja sigui de forma aleatòria o recomanada.

Gràcies a Spring i al patró de *Dependency Injection* és al fitxer applicationContext.xml on es defineixen els diferents tipus de tests, on se li diu a *TestGenerator* quines classes concretes ha d'agafar tant de *ListOfQuestions* com de *SelectionOfQuestions*:

```
<bean id="testGeneratorByObjective"
  class="org.innovacampus.utils.testgenerator.impl.TestGeneratorImpl">
  <property name="listOfQuestions" ref="listOfQuestionsByObjective" />
  <property name="selectionOfQuestions" ref="randomSelection"/>
  </bean>

  <bean id="testGeneratorByObjectiveAndDifficultyDegree"
    class="org.innovacampus.utils.testgenerator.impl.TestGeneratorImpl">
    <property name="listOfQuestions" ref="listOfQuestionsByObjectiveAndDifficultyDegree" />
    <property name="selectionOfQuestions" ref="randomSelection"/>
  </bean>
```

Generació de testos amb dificultat

Per ampliar la funcionalitat de l'aplicació hem afegit la possibilitat de generar testos segons el nivell de dificultat de les preguntes. La idea és que es pugui escollir entre testos fàcils, de nivell mig o difícils.

El problema fou que la dificultat és un terme una mica abstracte i volem que sigui un valor que es vagi ajustant automàticament. Per aconseguir-ho hem associat una dificultat a cada pregunta, i es recalcularà fent la mitjana de la dificultat que li ha suposat a cada estudiant respondre la pregunta en qüestió.

Els camps que es tenen en compte per a decidir quant ha costat resoldre la pregunta es té en compte:

- Respondre bé o malament la pregunta.
- Aprovar o suspendre el test.
- Temps que tarda en respondre la pregunta respecte al temps que tarda en respondre les altres preguntes del test.
- Temps que tarda en respondre una pregunta en relació al temps que han tardat la resta d'estudiants.

Per implementar aquesta part hem utilitzat lògica difosa [20] tecnologia que ens permet quantificar la vaguetat, i per tant poder ajustar la dificultat d'una forma més realitsta i precisa que fent-ho amb una sèrie de nivells preestablerts.

Seguidament explicarem la lògica difosa amb un exemple de velocitat. No es considera que un vehicle va ni prou ràpid o poc a poc, si no que es permet dir que va un 75% de velocitat normal i un 25% lenta. Per això és anomenada la brillant ciència dels grisos, perquè permet matisar, no és tot blanc o negre. I, segons aquestes premisses, en la gràfica s'obté que la velocitat pren el valor de 55.

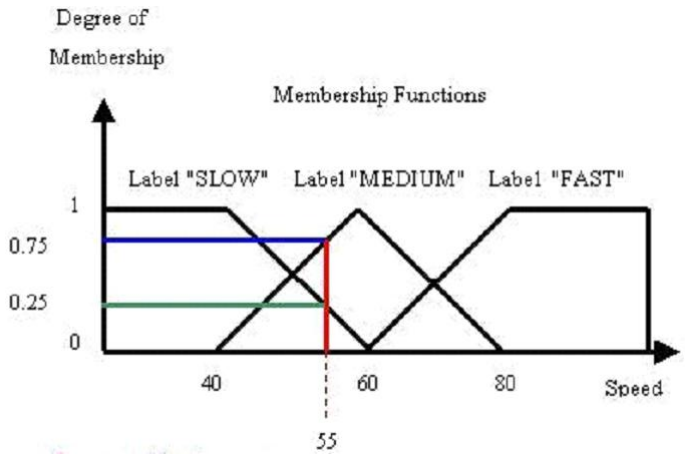

Figura 6.1: Gràfica exemple de lògica difosa.

En la implementació de la dificultat en les preguntes hem emprat la llibreria de lògica difosa JFuzzyLogic [16] que ens facilita el treballar amb conceptes difosos. A banda d'això ens redueix l'acoblament entre la lògica difosa i InnovaCampus ja que tota la configuració es troba en un fitxer extern.

6.3 Generació de gràfiques

La generació de gràfiques és una part d'InnovaCampus que formava part del mòdul d'estadístiques que es va integrar a InnovaCampus fa dos anys [3]. Nosaltres hem integrat tot el mòdul dins de l'aplicació i l'hem adaptat a Spring. Primer de tot veurem quin és el disseny de la generació de gràfiques i després veurem els canvis que s'han fet per adaptar-ho a Spring.

Per generar les gràfiques s'utilitza la biblioteca JFreeChart, que ofereix les classes necessàries per obtenir molts tipus de gràfiques totalment personalitzables. A més, permet exportar-les en forma d'imatges en els formats JPEG, PNG i SVG. També s'ha fet servir la biblioteca d'etiquetes Cewolf que ofereix etiquetes personalitzades per introduir gràfiques generades amb JFreeChart en pàgines JSP. Aquestes marques fan que les crides a les classes encarregades de generar les gràfiques i d'exportar-les siguin transparents per al programador.

Abans d'entrar en detalls sobre la implementació, hem de tenir clar què és un Dataset. Els dataset són els objectes utilitzats per les classes de la biblioteca JFreeChart. Aquests objectes contenen tota la informació a incloure: valors, títols, etiquetes, etc. Cada tipus gràfica ha de tenir el seu propi dataset.

La generació de gràfiques es va dissenyar i implementar fent servir el patró *Abstract-Factory* perquè fos poc repetitiva, reutilitzable i ampliable. Aquest patró s'utilitza quan hem de crear productes de famílies diferents, però per cada família tenim, més o menys, els mateixos productes. Llavors el client utilitza una fàbrica abstracta per crear qualsevol tipus de producte. Aquesta fàbrica abstracta serà superclasse de les factories concretes corresponents a cada una de les famílies. Seràn aquestes factories concretes les que s'encarreguin de crear els productes.

Tot i que en el patró original, la fàbrica abstracta defineix mètodes per crear cadascun dels productes de cada família, en el nostre cas només se'n defineix un i es deixa que siguin les fàbriques concretes les que crein els productes segons els paràmetres que se li passen. Amb els mètodes de factoria parametritzats flexibilitzem encara més la creació de productes ja que no és necessari que totes les famílies tinguin tots els productes. A més, l'addició de noves classes també resulta més fàcil.

Relació entre factories, productes i les nostres classes:

- Fàbrica abstracta: *CategoryDatasetProducerFactory*
- Fàbriques concretes:
  - *ObjectivesCategoryDatasetProducerFactory*: Factoria per a crear *DatasetProducers* relacionats amb la generació de gràfiques d'objectius.

  - *QuestionsCategoryDatasetProducerFactory*: Factoria per a crear *DatasetProducers* relacionats amb la generació de gràfiques de preguntes.
  - *TestsCategoryDatasetProducerFactory*: Factoria per a crear *DatasetProducers* relacionats amb la generació de gràfiques de tests.
  - *TopicsCategoryDatasetProducerFactory*: Factoria per a crear *DatasetProducers* relacionats amb la generació de gràfiques de temes.

- Famílies de productes i productes:
  - *MarksCategoryDatasetProducer*: Crea *DatasetProducers* per generar gràfiques on els valors seràn notes mitjanes.
    * *ObjectivesMarksCategoryDatasetProducer*
    * *QuestionsMarksCategoryDatasetProducer*
    * *TestsMarksCategoryDatasetProducer*
    * *TopicsMarksCategoryDatasetProducer*
  - *StudentsCategoryDatasetProducer*: Crea *DatasetProducers* per generar gràfiques on els valors seràn el nombre d'estudiants diferents que han realitzat tests per un tema.
    * *ObjectivesStudentsCategoryDatasetProducer*
    * *TopicsStudentsCategoryDatasetProducer*
  - *TestsCategoryDatasetProducer*: Crea *DatasetProducers* per generar gràfiques on els valors seràn el número de tests finalitzats i el número de tests aprovats.
    * *ObjectivesTestsCategoryDatasetProducer*
    * *TopicsTestsCategoryDatasetProducer*

La biblioteca JFreeChart ofereix classes per a crear multitud de gràfiques diferents que, conseqüentment hauran d'utilitzar *datasets*, també diferents. Les gràfiques de l'aplicació actual només utilitzen el que s'anomenen emphCategoryDatasets, que serveixen bàsicament per crear gràfics de barres i de línies. Per poder permetre que si es vol es puguin utilitzar altres tipus de gràfiques, com ara una gràfica de sectors, en un futur es van crear dues abstraccions més per sobre de les famílies de productes:

- *InnovaDatasetProducer*: Superclasse per a tots els *DatasetProducer* de l'aplicació. L'utilitzem per establir aspectes comuns que hauran de tenir totes les seves subclasses.

- *CategoryDatasetProducer*: *DatasetProducer* que creen *datasets* per a gràfics de barres i de línies.

També es va crear les abstraccions adients per damunt de la fràbrica abstracta:

- *DatasetProducerFactory*: Superclasse per a totes les factories de *DatasetProducer* de l'aplicació.

- *CategoryDatasetProducerFactory*: Factoria de *DatasetProducer* per a crear *datasets* per a gràfics de barres i de línies.

Per obtenir informació més detallada sobre els aspectes d'implementació de la generació de gràfiqes, consultar el TFC: *Disseny i implementació d'un mòdul d'estadístiques per una aplicació d'autoavaluació desenvolupada en comunitat* [3].

Adaptació a Spring

Com que hem fet un redisseny complert d'InnovaCampus, hem implementat de nou tota la aplicació. Com hem dit en el capítol 4, tota la funcionalitat de l'aplicació està dividida en subsistemes, i la informació necessària per generar les gràfiques l'obtindrem d'aquests subsistemes. Per tant, tots els productes de la generació de gràfiques (com *QuestionsMarksCategoryDatasetProducer*, *TestsMarksCategoryDatasetProducer*, *ObjectivesStudentsCategoryDatasetProducer* i *TopicsTestsCategoryDatasetProducer*) els hem adaptat a que obtinguin la informació dels subsistemes seguint la lògica de Spring i la injecció de dependència:

```
<bean id="testsMarksCategoryDatasetProducer"
  class="org.innovacampus.utils.datasetproducers.TestsMarksCategoryDatasetProducer">
  <property name="subjectManagement" ref="subjectManagement" />
  <property name="statisticsManagement" ref="statisticsManagement" />
</bean>
```

Fins a aquest punt l'adaptació a Spring era fàcil, ja que simplement vam resseguir el codi de les classes de productes i féiem que les crides a funcions que servien per obtenir informaicó cridessin a funcions que formaven part d'algun dels subsistemes.

El problema el vam tenir quan intentàvem generar gràfiques. Tal com estava implementat, se li indicava a una de les etiquetes de Cewolf (Biblioteca d'etiquetes que ofereix etiquetes personalitzades per introduir gràfiques generades amb JFreeChart en pàgines JSP) quina era la classe que s'encarregava de la generació de gràfiques, i així ja s'obtenia una instància de la classe i es generava el gràfic corresponent. Però com que ara els productes de la generació de gràfiques obtenien la informació dels subsistemes seguint la lògica de Spring i la injecció de dependència, és el contenedor de Spring qui li dóna una instància dels subsistemes als productes; i per tant si se li diu a Cewolf quina és la classe encarregada de la generació de gràfiques, la classe serà incapaç d'obtenir les instàncies dels subsistemes i llavors no es podrà generar cap gràfic.

Per solucionar-ho hem hagut de fer que sigui un controlador qui obtingui la classe generadora de gràfiques i li doni una instància d'aquesta classe a Cewolf. Com que el controlador forma part de Spring, llavors el contenidor de Spring sí que li donarà les instàncies dels subsistemes. D'aquesta manera Cewolf pot generar les gràfiques sense cap problema.

Tot seguit veurem el codi del controlador que s'encarrega d'obtenir la classe generadora de gràfiques. Aquest controlador és el *Chart.java* del paquet *org.innovacampus.mvc.controller.teacher*.

```
public class Chart extends AbstractController {

  private TopicsCategoryDatasetProducerFactory topicsCategoryDatasetProducerFactory;
  private ObjectivesCategoryDatasetProducerFactory objectivesCategoryDatasetProducerFactory;
  private QuestionsCategoryDatasetProducerFactory questionsCategoryDatasetProducerFactory;
  private TestsCategoryDatasetProducerFactory testsCategoryDatasetProducerFactory;
```

```
  [...]

  protected ModelAndView handleRequestInternal (HttpServletRequest request,
   HttpServletResponse response) throws Exception {

    ModelAndView mav = new ModelAndView("chart");

    String datasetType = request.getParameter("datasetType");
    String datasetProducer = request.getParameter("datasetProducer");

    InnovaDatasetProducer dataset = null;

    if(datasetProducer.equals("topicsDatasetProducer")){
      dataset = this.topicsCategoryDatasetProducerFactory.createDatasetProducer(datasetType);
      [...]
    }
    else if(datasetProducer.equals("objectivesDatasetProducer")){
      dataset = this.objectivesCategoryDatasetProducerFactory.createDatasetProducer(datasetType);
      [...]
    }
    else if(datasetProducer.equals("questionsDatasetProducer")){
      dataset = this.questionsCategoryDatasetProducerFactory.createDatasetProducer(datasetType);
      [...]
    }
    else if(datasetProducer.equals("testsDatasetProducer")){
      dataset = this.testsCategoryDatasetProducerFactory.createDatasetProducer(datasetType);
      [...]
    }

    mav.addObject("chartDataset",dataset);

    [...]
  }

  [...]
}
```

Com es pot veure al codi, el controdador crida a una fàbrica o a una altra depenent del valor del paràmetre *datasetProducer*. D'aquesta manera és el controlador qui s'encarrega d'obtenir una instància de la classe generadora de gràfiques i qui li passa a Cewolf a través de l'object *ModelAndView*.

Capítol 7

Proves de l'aplicació

El desenvolupament d'InnovaCampus ha sigut gradual. A grans trets podríem dir que primer vam començar amb la part de persistència, seguidament amb la part de serveis i després amb la part d'interfície. Com que anàvem avançant per parts, i com que el treballar amb una part implicava que l'anterior estigués acabada i funcionés correctament vam decidir anar fent proves per comprovar el bon funcionament de les classes i funcions de cada part. Nosaltres hem fet les proves de testeig de l'aplicació fent servir JUnit.

7.1 JUnit

JUnit és un conjunt de classes que permet realitzar l'execució de classes Java de manera controlada, per poder avaluar si el funcionament de cada un dels mètodes de les classes es comporta com s'espera. És a dir, en funció d'un valor d'entrada s'avalua el valor de retorn esperat; si la classe compleix amb l'especificació, llavors JUnit retornarà que el mètode de la classe ha passat amb èxit la prova; en cas de que el valor esperat sigui diferent al que ha retornat el mètode durant l'execució, JUnit retornarà un error en el mètode corresponent. Més informació [18, 19].

En el nostre cas, hem comprovat el correcte funcionament de les classes DAO, dels quatre subsistemes i de les classes generadores de tests. Per fer-ho d'una manera organitzada i estructurada hem creat una classe anomenada *RunJUnitTests* amb un mètode que crida a les classes: *RunJUnitDaoTests*, *RunJUnitServicesTests* i *RunJUnitTestGeneratorTests*. Cadascuna d'aquestes classes s'encarrega de cridar a totes les classes de testeig de les classes DAO, dels subsistemes (o classes de serveis) i de les generadores de tests respectivament. D'aquesta manera cridant a un sol mètode de la classe *RunJUnitTests* ja es comprova el correcte funcionament de totes les funcions de les diferents classes.

Per comprovar que els mètodes facin el que han de fer, hem testejat cadascun del mètodes comprovant tant l'execució normal com forçant una execució amb errors perquè saltin les excepcions esperades. Tot seguit mostrarem un exemple de la classe *AdministrationManagementTest* encarregada de comprovar el correcte funcionament del subsistema *AdministrationManagement*. En aquest exemple hi ha un primer mètode que insereix un professor nou al sistema i, un cop inserit, comprova que aquest nou professor o usuari existeixi. El segon mètode intenta inserir un altre cop al mateix professor, però com que aquest professor ja existeix saltarà l'excepció *TeacherExistsException*. L'etiqueta

@Test indica que el metòde s'ha de testjar i si es fica *@Test (expected = MyException.class)* indiques que en el testeig d'aquell mètode s'espera obtenir l'exepció *MyException*.

L'exemple:

```
@Test
public void test01_AddTeacher() throws TeacherExistsException {
  am.addTeacher(t.getName(),t.getSurnames(),t.getNIF(),t.getPassword(),
    t.getPhone(),t.getEmail());
  try{ assertNotNull(am.getUser(t.getNIF())); assertTrue(true);}
  catch (UserNotExistsException e){ fail("Teacher has not been added");}
}

@Test(expected = TeacherExistsException.class)
public void test02_AddTeacher_Existing()
  throws TeacherExistsException, UserNotExistsException {
  am.addTeacher(t.getName(),t.getSurnames(),t.getNIF(),t.getPassword(),
    t.getPhone(),t.getEmail());
}
```

Com que InnovaCampus està fet amb Spring, quan s'executa el testeig de les operacions amb JUnit la crida de java s'ha de fer utilitzant un agent de java, en el nostre cas la llibreria *spring-agent.jar*, indicant-li a java on es troba la llibreria. De manera que la crida quedaria de la següent manera:

```
>java -javaagent:spring-agent.jar org.innovacampus.test.RunJUnitTests
```

Capítol 8

Model de desenvolupament

8.1 Model de desenvolupament

El model de desenvolupament a seguir a partir del punt en que publiquem aquest prototipus prometedor serà treballar sobre el repositori sedna http://sedna.udl.cat/svnInnovaSp, sense deixar de banda el gestor de versions Subversion i tenint present que InnovaCampus és una aplicació que es troba sota la Llicència GPL.

Un punt crític és decidir quan una modificació està preparada per a ser publicada. Doncs és interessant intentar alliberar ràpid i sovint, en mòduls petits i delegant tasques al màxim possible. No és necessari esperar a tenir mòduls complerts per a publicar-ho, és bo anar penjant les modificacions i sobretot presentar-les d'una forma atractiva per incentivar nous desenvolupadors a continuar amb aquesta feina.

8.2 Ajuts al desplegament i al desenvolupament

Aquest capítol va destinat a aquelles persones que continuaran el projecte InnovaCampus després de nosaltres, per aquells que continuaran la nostra feina. Tot seguit descriuriem els passos que hauria de seguir una persona nova que entrés a formar part de l'equip de treball.

Primer de tot necessitaria accedir a l'interior d'InonvaCampus, per poder veure tot el que hi ha fet fins ara, el codi font, fitxers de configuració... Això és molt fàcil, ja que tota l'aplicació està disponible en un repositori. Aquest repositori és accesible des d'Internet i tothom qui vulgui pot veure el seu contingut.

En el repositori hi ha instal·lat un sistema de control de versions, Subversion (svn) A.8. D'aquesta manera, totes les modificacions en l'aplicació queden enregistrades pel gestor i tots els membres de l'equip de treball poden veure les modificacions que s'han fet.

Si el nou membre de l'equip volgués modificar l'aplicació necessitaria el codi font per poder fer els canvis oportuns. Abans ja hem dit que es troba en el repositori, però per poder descarregar l'aplicació i poder-hi fer modificacions, el nou membre hauria de conectar-se al repositori mitjançant l'eina de subversion. D'aquesta manera, i tenint el permisos corresponents, podria obtenir tots els fitxers que necessités, fer les modificacions

que calguessin i tornar-ho a col·locar al repositori mitjançant subversion. Així el gestor de versions enregistraria una nova actualització.

Un altre aspecte a tenir un compte és la compilació. Com que és una aplicació tan gran, hi ha fitxers de configuració, de propietats, les pàgines web, les fulles d'estils, imatges, el codi font, documents javascript... i tot està ordenat i agrupat en directoris. Per poder-ho compilar i que trobi tots els fitxers, fem servir l'eina Ant. Apache Ant és una eina usada en programació per a la automatització de tasques mecàniques i repetitives, normalment durant la fase de compilació i construcció (build). En un fitxer anomenat build.xml es defineix el procés de construcció.

Fent servir l'eina Ant es permet que la persona que vulgui compilar l'aplicació pugui fer-ho directament o pugui utilitzar un IDE(Un entorn integrat de desenvolupament o IDE, és una eina informàtica per al desenvolupament de programari de manera còmoda i ràpida) com NetBeans o Eclipse. Així si el nou membre de l'equip està acostumat a fer servir NetBeans, es pot instal·lar un mòdul que permet utilitzar subversion dins de NetBeans. Amb eclipse també hi ha un mòdul que et pemet fer servir subversion.

Per poder executar l'aplicació s'ha de fer més coses a bandad de descarregar-la i compilar-la. S'ha de crear la base de dades, amb totes les taules per poder-hi guardar la informació, i s'han d'incloure algunes llibreries a Tomcat. En l'apartat de documentació dins del repositori hi ha un fitxer anomenat "deployment.txt"on s'explica detalladament els passos a seguir per a desplegar l'aplicació, així com el lloc on es troba el fitxer sql de creació de la base de dades i les llibreries que s'hauran de ficar a Tomcat. Així doncs, si el nou membre de l'equip volgués provar l'aplicació se la descarregaria, seguiria els passos per fer el desplegament, executaria el build.xml i ja se li compilaria, se li col·locaria en el directori corresponent de Tomcat i ja estaria preparada per provar-la.

Fins ara hem parlat de com obtenir l'aplicació, que la podrem modificar, com compilar-la, llavors la podrem provar, i que per mitja de subversion actualitzarem l'aplicació del repositori. Però quan es modifica o es crea qualsevol cosa en l'aplicació ha d'estar ben documentat. Aquesta és una de les parts més importants, ja que nosaltres no serem els únics que treballarem sobre l'aplicació i hi haurà més persones que continuaran la nostra feina. És per això que hem explicat com està estructurat el repositori, els passos a seguir per poder desplegar l'aplicació, hem inclòs el model de domini i el de la base de dades, i hem explicat mitjançant javadoc quina és la finalitat i per a que serveixen les classes i les seves funcions. Si el nou membre de l'equip dotés de nova funcionalitat a l'aplicació, quan creés les noves classes i funcions hauria d'incloure-les al javadoc, així tots els membres de l'equip i les noves incorporacions podrien fàcilment entrendre per a que serveixen les classes i la finalitat de cada funció.

El nou membre de l'equip de treball s'encarregaria d'ampliar la funcionalitat de l'aplicació, però no ho faria a nivell global sinó més en particular, és a dir, InnovaCampus és una aplicació molt gran i intentar-hi treballar a nivell global és bastant complicat i farragós, és millor veure l'aplicació com dividida en diferents parts o mòduls; tot seguit en citarem uns quants.

El mòdul de la generació de tests, que s'encarrega dels tipus de tests i com es generen.

El mòdul de recursos, les preguntes d'un test poden tenir associat un recurs i aquest mòdul gestiona els tipus de recursos que hi ha (audio, video i document) i la manera com es visualitzaran.

El mòdul d'estadístiques, és el responsable de la generació de les gràfiques que visualitzarà el professor per poder fer el seguiment dels seus alumnes.

El mòdul d'exportació de dades, en aquest mòdul podem incloure tant l'exportació de tota la informació d'una assignatura com l'exportació de les gràfiques de les estadístiques en diferents formats (pdf, excel, txt).

El mòdul de la interfície web, aquí fem referència a les vistes i el disseny de les pàgines web. Per exemple els colors que s'utilitzen, el lloc on hi ha cada element o el tipus de lletra.

En resum, treballar en InnovaCampus implica treballar en equip i per tant s'utilitzarà un gestor de versions per tenir un major control i seguretat en el progrés de l'aplicació, ha d'estar tot ben documentat ja que no serem els únics que hi treballarem i tothom ha de poder saber per a que serveixen les noves operacions que creem, i que InnovaCampus és una aplicació molt gran que la podem dividir en diferents mòduls.

Capítol 9

Conclusions i treball futur

En aquest projecte hem fet un redisseny de l'aplicació InnovaCampus per a convertir-la en un prototip prometedor i poder-la sotmetre així al desenvolupament en comunitat i modular. A nivell de millores en l'enginyeria del software hem emprat els frameworks Spring i JPA, hem aplicat els patrons MVC i DAO, i n'hem obtingut una arquitectura de tres capes on les capes estan completament desacoblades entre sí. I en quant al model Bazar de desenvolupament en comunitat hem incorporat un gestor de versions, un repositori, i ho hem documentat tot a ,fons tant per a poder desplegar l'aplicació com per a poder-la ampliar d'una forma senzilla.

Aquest projecte ens ha fet partíceps del món InnovaCampus, on hem estat continuant la tasca d'altres desenvolupadors i treballant col·laborativament utilitzant un gestor de versions, documentant totes i cadascuna de les classes amb javadoc i elaborant fitxers com el 'readme', 'desplegament de l'aplicació' i 'estructura' que permetran als futurs desenvolupadors sentir-se còmodes i familiaritzar-se amb el projecte des del primer moment. Això ens dóna la tranquil·litat d'haver fet la feina ben feta, de que realment desenvolupar una aplicació no és només escriure línies de codi.

A banda d'intentar aplanar el camí als nostres companys també hem intentat divertir-nos. Un dels punts del projecte en els que més vàrem disfrutar fou treballant amb intel·ligència artificial, tant per aplicar dificultat a les preguntes com per a generar testos recomanats. Aquest és un àmbit en el que no havíem aprofundit mai i que ens ha ofert un gran ventall de possibilitats per donar valor afegit a aquesta eina d'autoavaluació.

No tot ha estat flors i violes, al principi, quan se'ns va comentar que hauríem d'utilitzar Spring i JPA ens va caure el món a sobre, perquè no ho havíem utilitzat mai i perquè a en ser tecnologies tan recents ens resultà molt difícil trobar manuals i tutorials amb cara i ulls. Poc a poc ens hi vam anar familiaritzant, i a dia d'avui ja podem afegir, amb orgull, als nostres currículums que tenim coneixements i pràctica en aquest camp.

Finalment, i seguint amb la tònica d'optimisme i satisfacció que ens aporta finalitzar, amb èxit segons els objectius plantejats, el nostre granet de sorra en aquest projecte, només dir que desenvolupar seguint la metodologia correcta (anàlisi de requisits, disseny de la base de dades, disseny del model de domini, classes DAO i classes de serveis amb les respectives interfícies), és a dir, implementant una arquitectura de tres capes fidelment

a la seva naturalesa, ens fa sentir que hem estat capaços de realitzar un projecte digne d'una enginyeria informàtica.

Com a aspectes a millorar i treball futur creiem que seria interessant:

- Acabar d'implementar la interfície
- Crear nous tipus de testos
- Permetre adjuntar fórmules i fragments de codi a les preguntes
- Oferir més funcionalitats als estudiants per poder personalitzar-se l'aplicació
- Afegir un fòrum per comentar dubtes

Apèndix A

Tecnologies utilitzades

A.1 Java

El llenguatge de programació Java fou dissenyat per James Gosling i els seus companys a Sun Microsystems, a l'any 1990, a partir del C++. Des del seu naixement fou pensat com un llenguatge orientat a objectes, és a dir, que segueix la filosofia de programar mòduls senzills, per tal de crear aplicacions avançades quan tots treballen junts.

A quest és un llenguatge interpretat i, per tant, pot semblar lent en comparació amb altres llenguatges, però ofereix un índex de reutilització del codi molt elevat i és possible de trobar moltes llibreries lliures de Java. És un llenguatge flexible i potent per la facilitat amb què es programa i els resultats que ofereix. Un dels trets que el caracteritza i que fa que sigui una eina molt valorada a l'hora de desenvolupar aplicacions distribuïdes, és el fet que sigui un llenguatge multi-plataforma. Per més informació [13, 14].

A.2 Spring

El Spring Framework (també conegut simplement com Spring) és un frameword de codi obert de desenvolupament d'aplicacions per la plataforma Java i per .NET Framework [25]. Per més informació sobre Spring [23, 24, 5].

Les característiques fundamentals del framework Spring es poden utilitzar per qualsevols aplicació Java, però hi ha mòduls per crear aplicacions web per damunt de la plataforma Java Enterprise. Encara que Spring no imposa un model de programació específic, s'ha fet popular en la comunitat Java com una alternativa, substitució o afegit del model de Enterprise JavaBean (EJB).

Spring és un contenedor i marc de treball lleuger d'injecció de dependencia i orientat a aspectes. Això és molt, però resumeix bé el proposit principal de Spring. Per poder-ho entendre més bé, desglosem aquesta descripció:

- Contenedor: Spring és un contenedor en el sentit de que conté i gestiona el cicle de vida i configuració del objectes de l'aplicació.
- Marc de treball: Spring fa possible configurar i escriure aplicacions complexes a partir de components senzills.

- Lleuger: Spring és lleuger tant en termes de tamany com en temps de processament.
- Injecció de dependencia: Spring fomenta el baix acoplament mitjançant una tècnica coneguda com injecció de dependencia (DI). Quan s'aplica la DI, s'otorga als objectes de forma pasiva les seves dependencies, en lloc de crear o buscar objectes dependents entre ells. En lloc de que un objecte busqui dependencies en un contenedor, és el contenedor qui otorga les dependencies al objecte en la creació de les instancies sense esperar a que ho demani.
- Orientat a aspectes: Spring té un ample soport per a la programació orientada a aspectes (AOP) que permet el desenvolupament cohesiu separan la lògica empresarial de l'aplicació dels serveis del sistema (com auditoria i gestió de transaccions).

A.3 Spring Security

Spring Security és un framework de Java/Java EE que proporciona autentiació abançada, autorització i altres aspectes de seguretat per aplicacions fetes amb Spring (Java). Per més informació [26].

L'autenticació és el procés de reconeixer que un usuari es qui diu ser. L'autorització es refereix al proces de decidir si un usuari està autoritzat a fer una acció en l'aplicació. Per arrivar al punt en que es necessaria una desició d'autorització, la identitat de l'usuari ja ha hagut de ser establerta pel proces d'autentiació.

A.4 JPA

Java Persistence API (JPA) és un marc de treball per a Java que permet als programadors gestionar dades relacionals en apliacions que fan servir la plataforma Java. Per més informació [17].

La persistència consisteix en tres àrees:

- L'API, definida al paquet javax.persistence.
- El Java Persistence Query Language.
- Les metadades objecte/relacional.

A.5 Ajax

AJAX (Asynchronous JavaScript And XML) és una tècnica de desenvolupament web per crear aplicacions interactives. Aquestes aplicacions s'executen en el client, és a dir, en el navegaddor de l'usuari mentres es manté una comunicació asíncrona amb el servidor en segon pla. D'aquesta manera es poden fer canvis sobre les pàgines sense necessitat de recargar-les, el que fa que augmenti la interactivitat, velocitat i usabilitat de les aplicacions.

En el nostre cas hem utilitzat el projecte AJAX Taglib, que és un conjunt d'etiquetes JSP (JavaServer Pages) que simplifiquen l'ús de la tecnologia AJAX en pàgines JSP. Per més informació [7].

A.6 jFuzzyLogic

jFuzzyLogic és un paquet de lògica difusa escrit en Java. Implementa el Fuzzy Control Language (FCL) especificació (IEC 1131p7) [12]. Per més informació sobre jFuzzyLofic [16].

La lògica difusa es basa en lo relatiu de l'observat. Aquest tipus de lògica agafa dos valors aleatoris, però contextualitzats i referenciats entre si. Així, per exemple, una persona que medeixi 2 metros és clarament una persona alta, si previament s'ha obtingut el valor de persona baixa i s'ha establert a 1 metro. Els dos valors estan contextualitzats a persones i referenciats a una mesura mètrica lineal. [20].

A.7 Cewolf

Cewolf és una biblioteca d'etiquetes que ens permet inserir imatges amb gràfiques generades amb JFreeChart. [3, 15, 9].

Es pot fer servir dins d'aplicacions basades en Servlet/JSP per insertar gràfics complexos de tot tipus (lineals, circulars, de barres, histogrames...) en una pàgina web. S'utilitza fent servir una completa llibreria d'etiquetes que permet definir totes les propietats del gràfic (color, ralles, llegendes...). D'aquesta manera el JSP que inclou el gràfic no queda ple de codi Java, tot es descriu amb etiquetes XML.

A.8 SVN

Subversion (SVN) és un sistema de control de versions de codi obert. Gestiona fitxers i directoris, i els canvis que es produeixen en ells en el temps. Això li permet recuperar versions antigues de les dades o veure l'historial de com han anat canviant les dades. En aquest sentit, moltes persones comparen un sistema de control de versions com una "màquina del temps". Per més informació [27].

Apèndix B

Llicència GPL

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. http://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

To protect your rights, we need to prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that

modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

Terms and Conditions

0. Definitions.

   "This License" refers to version 3 of the GNU General Public License.

   "Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

   "The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

   To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

   A "covered work" means either the unmodified Program or a work based on the Program.

   To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

   To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

   An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

   The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

   A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

   The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

   The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

   The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

   The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

   All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

   You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

   No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

   When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

   You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

   You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

   You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

   (a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

   (b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

   (c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

   (d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

   A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined

with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

   You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

   (a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

   (b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

   (c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

   (d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

   (e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

   A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

   A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining

whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered

work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

(a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

(b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

(c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

(d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

(e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

(f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is

the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

   You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

    Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

    An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

    You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

    A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

    A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of

this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from

the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

    Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

    The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

    Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

    If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

    Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

    THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

    IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO

MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

    If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

End of Terms and Conditions

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>

Copyright (C) <textyear>  <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

Also add information on how to contact you by electronic and paper mail.

If the program does terminal interaction, make it output a short notice like this when it starts in an interactive mode:

```
<program>  Copyright (C) <year>  <name of author>

This program comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands `show w` and `show c` should show the appropriate parts of the General Public License. Of course, your program's commands might be different; for a GUI interface, you would use an "about box".

You should also get your employer (if you work as a programmer) or school, if any, to sign a "copyright disclaimer" for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see `http://www.gnu.org/licenses/`.

The GNU General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Lesser General Public License instead of this License. But first, please read `http://www.gnu.org/philosophy/why-not-lgpl.html`.

Apèndix C

Javadoc

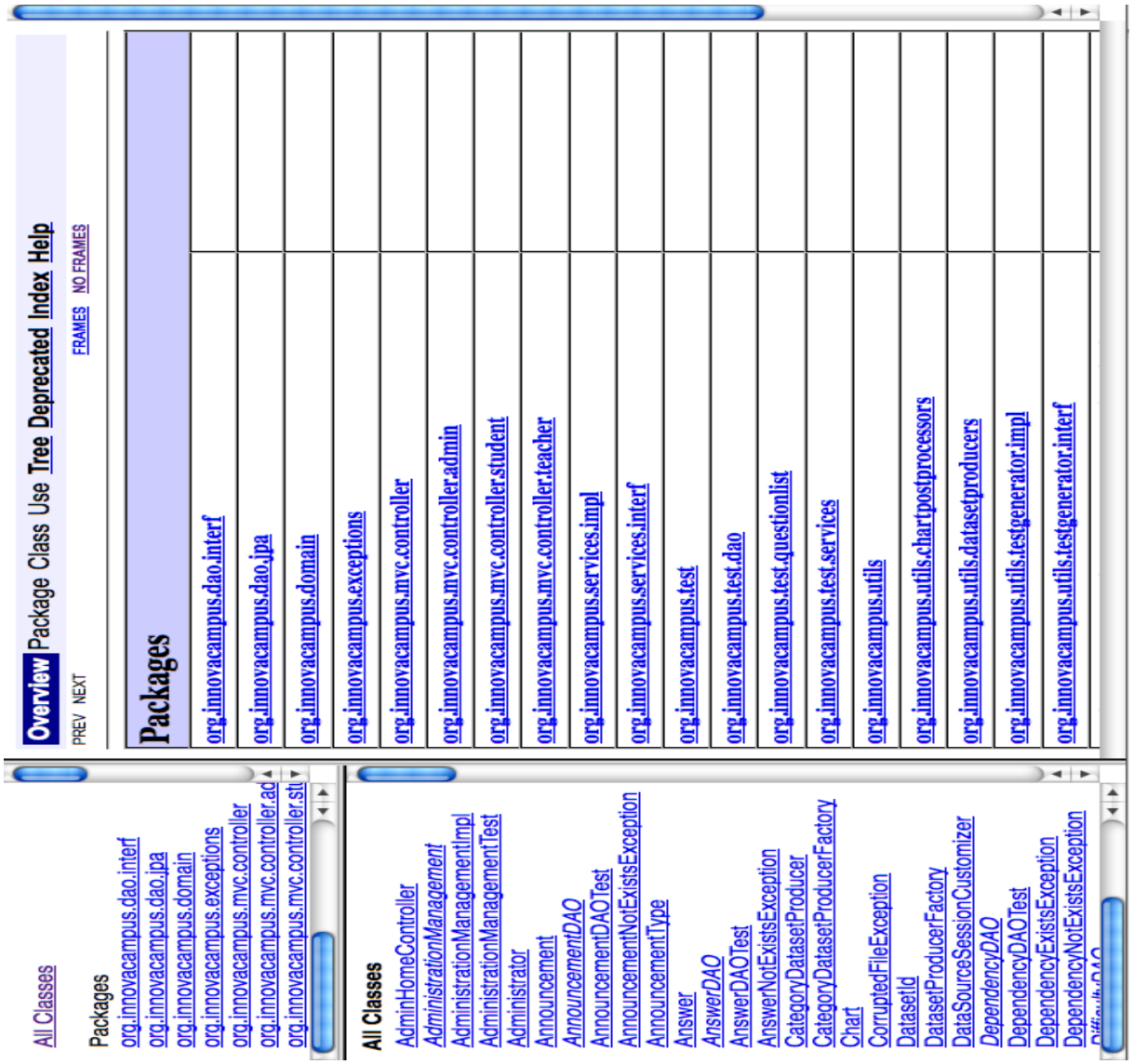

Figura C.1: Menú principal amb tots els paquets

All Classes

Packages
org.innovacampus.dao.interf
org.innovacampus.dao.jpa
org.innovacampus.domain
org.innovacampus.exceptions
org.innovacampus.mvc.controller
org.innovacampus.mvc.controller.ad
org.innovacampus.mvc.controller.st

All Classes
AdminHomeController
AdministrationManagement
AdministrationManagementImpl
AdministrationManagementTest
Administrator
Announcement
AnnouncementDAO
AnnouncementDAOTest
AnnouncementNotExistsException
AnnouncementType
Answer
AnswerDAO
AnswerDAOTest
AnswerNotExistsException
CategoryDatasetProducer
CategoryDatasetProducerFactory
Chart
CorruptedFileException
DatasetId
DatasetProducerFactory
DataSourceSessionCustomizer
DependencyDAO
DependencyDAOTest
DependencyExistsException
DependencyNotExistsException

Overview Package Class Use Tree Deprecated Index Help

PREV PACKAGE NEXT PACKAGE FRAMES NO FRAMES

Package org.innovacampus.utils.datasetproducers

Class Summary

CategoryDatasetProducer	A **CategoryDatasetProducer** produces the kind of datasets needed to create barcharts, etc.
CategoryDatasetProducerFactory	Superclass for all the CategoryDatasetProducer factories
DatasetId	The public static attributes from a **DatasetId** are used to identify a kind of chart.
DatasetProducerFactory	**DatasetProducerFactory**'s are those classes which create DatasetProducers.
InnovaDatasetProducer	Superclass for all the DatasetProducers.
MarksCategoryDatasetProducer	MarksCategoryDatasetProducer base class.
ObjectivesCategoryDatasetProducerFactory	This Factory creates CategoryDatasetProducers to generate charts related to objectives.
ObjectivesMarksCategoryDatasetProducer	Creates a CategoryDatasetProducer filled with data related to objectives' marks
ObjectivesStudentsCategoryDatasetProducer	Creates a CategoryDatasetProducer filled with data related to objectives' students
ObjectivesTestsCategoryDatasetProducer	Creates a CategoryDatasetProducer filled with data related to objectives' tests.
QuestionsCategoryDatasetProducerFactory	This Factory creates CategoryDatasetProducers to generate charts related to questions.
QuestionsMarksCategoryDatasetProducer	Creates a CategoryDatasetProducer filled with data related to questions' marks.
StudentsCategoryDatasetProducer	StudentsCategoryDatasetProducer base class.
TestsCategoryDatasetProducer	TestsCategoryDatasetProducer base class.
TestsCategoryDatasetProducerFactory	This Factory creates CategoryDatasetProducers to generate charts related to tests.
TestsMarksCategoryDatasetProducer	Creates a CategoryDatasetProducer filled with data related to tests' marks
TopicsCategoryDatasetProducerFactory	This Factory creates CategoryDatasetProducers to generate charts related to topics.

Figura C.2: Paquet datasetproducers

Figura C.3: Interfície d'AdministratorManagement

Exemple del Javadoc generat per les funcions de la interfície AdministrationManagement, encarregada de la part més administrativa:

getUser

User getUser(java.lang.String userId) throws UserNotExistsException

Get a user

Parameters:

userId - user's id

Returns:

the user

Throws:

UserNotExistsException - if the user doesn't exist

getSubject

Subject getSubject(java.lang.String subjectId) throws SubjectNotExistsException

Get a subject

Parameters:

subjectId - subject's id

Returns:

the subject

Throws:

SubjectNotExistsException - if the subject doesn't exist

getAdministratorAnnouncementsById

Announcement getAdministratorAnnouncementsById(int id) throws AnnouncementNotExistsException

Get an announcement

Parameters:

id - announcement's id

Returns:

the announcement

Throws:

AnnouncementNotExistsException - if the announcement is not in the system

getSubjectAnnouncementById

SubjectAnnouncement getSubjectAnnouncementById(int id) throws SubjectAnnouncementNotExistsException

Get an announcement of a subject

Parameters:

id - announcement's id

Returns:

the announcement of the subject

Throws:

SubjectAnnouncementNotExistsException - if the announcement of the subject is not in the system

changePassword

void changePassword(java.lang.String userId, java.lang.String newPass) throws UserNotExistsException

Change the password of the user with userId

Parameters:

userId - user's id

newPass - new password

Throws:

UserNotExistsException - if this userId is not in system

rememberPassword

void rememberPassword(java.lang.String userId) throws UserNotExistsException, InnovaMailException

Generate a new password for this user, and return this one.

Parameters:

userId - user's id

Throws:

UserNotExistsException - if this userId is not in system

InnovaMailException - if it is not possible to send a mail

updateStudentData

void updateStudentData(java.lang.String userId, java.lang.String name, java.lang.String surnames, java.lang.String email) throws UserNotExistsException

Update student's data

Parameters:

userId - student's id

name - student's name

surnames - student's surnames

email - student's email

Throws:

UserNotExistsException - if this userId is not in system

updateTeacherData

void updateTeacherData(java.lang.String userId, java.lang.String name, java.lang.String surnames, java.lang.String email, java.lang.String phone) throws UserNotExistsException

Update teacher's data

Parameters:

userId - teacher's id

name - teacher's name

surnames - teacher's surnames

email - teacher's email

phone - teacher's phone

Throws:

UserNotExistsException - if this userId is not in system

signUp

void signUp(java.lang.String userId, java.lang.String email, java.lang.String password) throws UserNotExistsException, UserAlreadySignUpException

Sign up a user

Parameters:

userId - user's id

password - user's password

Throws:

UserNotExistsException - if it is a teacher that is because it has not been introduced in system if it is a student that is because it has not been added in any subject

IncorrectPasswordException - if the password is incorrect

UserAlreadySignUpException - if the user signs up before

signIn

void signIn(java.lang.String userId, java.lang.String password) throws UserNotExistsException, UserNotSignUpException, IncorrectPasswordException

Sign in a user

Parameters:

userId - user's id

password - user's password

Throws:

UserNotExistsException - if it is a teacher that is because it has not been introduced in system if it is a student that is because it has not been added in any subject

UserNotSignUpException - if the user is not signed up

IncorrectPasswordException - if the password is incorrect

getAdministratorAnnouncementsByUser

java.util.List getAdministratorAnnouncementsByUser(java.lang.String userId) throws UserNotExistsException

Get Administrator's announcements depending on type of user (teacher or student)

Parameters:

userId - user's id

Returns:

a list of administrator announcements

Throws:

UserNotExistsException - if this userId is not in system

getSubjectAnnouncementsBySubject

java.util.List getSubjectAnnouncementsBySubject(java.lang.String subjectId)
throws SubjectNotExistsException

Get a list of subject's announcements

Parameters:

subjectId - subject's id

Returns:

a list of subject's announcements

Throws:

SubjectNotExistsException - if this subject is not in system

addTeacher

void addTeacher(java.lang.String name, java.lang.String surnames, java.lang.String nif, java.lang.String password, java.lang.String phone, java.lang.String email)
throws TeacherExistsException

Add a teacher

Parameters:

name - teacher's name
surnames - teacher's surnames
nif - teacher's nif
password - teacher's password
phone - teacher's phone
email - teacher's email

Throws:

TeacherExistsException - if this teacher has already been introduced in the system

deleteTeacher

void deleteTeacher(java.lang.String userId) throws UserNotExistsException

Delete a teacher

Parameters:

userId - teacher's id

Throws:

UserNotExistsException - if this userId is not in the system

addAdministratorAnnouncement

void addAdministratorAnnouncement(java.lang.String text, AnnouncementType type)
XXXXXXXXX

Add an administrator's announcement

Parameters:

text - announcement's text
type - announcement's type

Throws:

IncorrectAnnouncementTypeException

deleteAdministratorAnnouncement

void deleteAdministratorAnnouncement(int announcementId) throws AnnouncementNotExistsException

Delete an administrator's announcement

Parameters:

announcementId - announcement's id

Throws:

AnnouncementNotExistsException - if this announcement's id doesn't exist in the system

getAdministratorAnnouncementsForUnloggedUsers

java.util.List getAdministratorAnnouncementsForUnloggedUsers()

Get Administrator's announcements for unlogged users

Returns:

a list of administrator announcements for unlogged users

addSubject

void addSubject(java.lang.String subjectId, java.lang.String name, java.lang.String description) throws SubjectExistsException

Add a subject

Parameters:

subjectId - subject's id
name - subject's name
description - subject's description

Throws:

SubjectExistsException - if this subject's id has already been introduced in the system

deleteSubject

void deleteSubject(java.lang.String subjectId) throws SubjectNotExistsException

Delete a subject

Parameters:

subjectId - subject's id

Throws:

SubjectNotExistsException - if this subject's id is not in the system

updateSubject

void updateSubject(java.lang.String subjectId, java.lang.String name, java.lang.String description) throws SubjectNotExistsException

Update subject's data

Parameters:

subjectId - subject's id
name - subject's name
description - subject's description

Throws:

SubjectNotExistsException - if this subject's id is not in the system

addResponsibleTeacherToSubject

void addResponsibleTeacherToSubject(java.lang.String subjectId, java.lang.String userId)
throws SubjectNotExistsException, UserNotExistsException,
TeacherAlreadyResponsibleException

Add a responsible teacher to a subject

Parameters:

subjectId - subject's id
userId - teacher's id

Throws:

SubjectNotExistsException - if this subject's id is not in the system
UserNotExistsException - if this user's id is not in the system
TeacherAlreadyResponsibleException - if this teacher was responsible of this subject

deleteTeacherFromSubject

void deleteTeacherFromSubject(java.lang.String subjectId,java.lang.String userId)
throws SubjectNotExistsException, UserNotExistsException,
TeacherNotResponsibleException

Delete a teacher from a subject

Parameters:

subjectId - subject's id
userId - user's id

Throws:

SubjectNotExistsException - if this teacher is not responsible of this subject
UserNotExistsException - if this user's id is not in the system
TeacherNotResponsibleException - if this user's id is not a responsible teacher of this subject

enrolStudentToSubject

void enrolStudentToSubject(java.lang.String userId, java.lang.String name,
java.lang.String surnames, java.lang.String subjectId)
throws SubjectNotExistsException, StudentAlreadyEnroledException

Enrol a student to a subject, it doesn't matter if this student was in the system or not.

Parameters:

userId - student's id
name - student's name
surnames - student's surnames
subjectId - subject's id

Throws:

SubjectNotExistsException - if this subject is not in the system
StudentAlreadyEnroledException - if this student is already enroled in this subject

enrolListOfStudentsToSubject

void enrolListOfStudentsToSubject(java.util.List students, java.lang.String subjectId) throws SubjectNotExistsException, StudentAlreadyEnroledException

Enrol a list of students to a subject

Parameters:

students - a list of students

subjectId - subject's id

Throws:

SubjectNotExistsException - if this subject is not in the system

StudentAlreadyEnroledException - if this student is already enroled in this subject

enrolStudentsByCSVFileToSubject

void enrolStudentsByCSVFileToSubject(java.lang.String filename, java.lang.String subjectId) throws FileNotFoundException, CorruptedFileException, SubjectNotExistsException, StudentAlreadyEnroledException

Enrol students from a CSV file to a subject

Parameters:

filename - the name of the file

subjectId - subject's id

Throws:

FileNotFoundException - if the file doesn't exist

CorruptedFileException - if this file is unreadable

SubjectNotExistsException - if this subject is not in the system

StudentAlreadyEnroledException - if this student is already enroled in this subject

deleteStudentFromSubject

void deleteStudentFromSubject(java.lang.String subjectId, java.lang.String userId) throws SubjectNotExistsException, UserNotExistsException, StudentNotEnrolledException

Delete a student from a subject

Parameters:

subjectId - subject's id

userId - student's id

Throws:

SubjectNotExistsException - if this subject is not in the system

UserNotExistsException - if the student doesn't exist

StudentNotEnrolledException - if this student is not enrolled in this subject

addSubjectAnnouncement

void addSubjectAnnouncement(java.lang.String subjectId, java.lang.String text) throws SubjectNotExistsException

Add a subject announcement

Parameters:

subjectId - subject's id

text - announcement's text

Throws:

SubjectNotExistsException - if this subject is not in the system

deleteSubjectAnnouncement

void deleteSubjectAnnouncement(int subjectAnnouncementId)
throws SubjectAnnouncementNotExistsException

Delete a subject announcement

Parameters:

subjectAnnouncementId - subject announcement's id

Throws:

SubjectAnnouncementNotExistsException - if this subject announcement is not in the system

Apèndix D

Casos d'ús de les classes service

D.1 Casos d'ús d'AdministrationManagement

Cas d'ús:	**Canviar el password d'usuari**
Descripció:	L'objectiu és que l'usuari pugui canviar el seu password i el sistema ho registri.
Actors:	Sistema i Professor o Sistema i Alumne
Precondicions:	L'usuari ha fet login a InnovaCampus i indica que es vol canviar el password.
Postcondicions:	El sistema ha registrat el nou password.

Flux principal:	
Usuari	**Sistema**
1. L'usuari indica que vol que vol canviar el seu password.	2. El Sistema li proporciona el formulari.
3. L'usuari introdueix el nou password	4. El Sistema informa que el canvi ha estat satisfactori.

Casos d'error:
3. La contrasenya no és vàlida. No s'efectuarà el canvi i s'informarà de l'error.

Cas d'ús:	**Recordar clau**
Descripció:	L'objectiu és que l'usuari rebi una clau per accedir a InnovaCampus.
Actors:	Sistema i Professor o Sistema i Alumne.
Precondicions:	L'usuari no ha fet login a InnovaCampus.
Postcondicions:	El sistema ha enviat la nova clau al correu de l'usuari.

Flux principal:	
Usuari	**Sistema**
1. L'usuari indica que vol que vol recordar el password.	2. El Sistema genera una nova clau, l'emmagatzema per aquell usuari i li envia al seu correu.
	3. El Sistema informa que l'enviament ha estat satisfactori.

Cas d'ús:	**Modificar les dades de l'usuari**
Descripció:	L'objectiu és que l'usuari pugui modificar les seves dades personals, com el nom, els cognoms, el mail i en cas del professor també el telefon.
Actors:	Sistema i Professor o Sistema i Alumne
Precondicions:	L'usuari ha fet login a InnovaCampus i indica que vol canvia les seves dades.
Postcondicions:	El sistema enregistra les modificacions al sistema.

Flux principal:	
Usuari	**Sistema**
1. L'usuari indica que vol modificar les seves dades	2. El sistema li proporciona el formulari amb les dades actuals
3. L'usuari fa les modificacions que desitja	4. El sistema guarda els canvis i ho notifica a l'usuari.

Casos d'error:
3. Alguna de les dades introduïdes pot ser incorrecta. L'usuari queda amb les dades anteriors i s'informa de l'error.

Cas d'ús:	**Iniciar sessió**
Descripció:	L'objectiu és que l'usuari iniciï sessió a InnovaCampus
Actors:	Sistema i Professor o Sistema i Alumne
Precondicions:	L'usuari vol iniciar sessió dins l'aplicació
Postcondicions:	L'usuari inicia sessió a l'aplicació

Flux principal:	
Usuari	**Sistema**
1. L'usauri vol iniciar sessió a InonvaCampus.	2. El sistema li mostra la pàgina d'inici amb el formulari d'entrada.
3. L'usuari introdueix el seu identificador i contrasenya.	4. El sistema comprova les dades i li dona accés a l'aplicacó.

Casos d'error:
3. L'usuari encara no està donat d'alta, no se li dóna accés i s'informa de l'error. 3. L'usuari encara no està registrat, no se li dóna accés i s'informa de l'error. 3. El identificador o contrasenya són incorrrectes, , no se li dóna accés i s'informa de l'error.

Cas d'ús:	**Registrar-se**
Descripció:	L'objectiu és que l'usuari és registri al sistema.
Actors:	Sistema i Professor o Sistema i Alumne
Precondicions:	El professor ha estat donat d'alta per l'administrador i l'alumne ha estat matriculat en alguna de les assignatures del sistema.
Postcondicions:	L'usuari ja està registrat al sistema i pot accedir al contigut de l'eina InnovaCampus.

Flux principal:	
Usuari	**Sistema**
1. L'usuari indica que vol registrar-se.	2. El sistema demana a l'usuari la seva identificació, correu electrònic i contrasenya.
3. L'usuari introdueix les dades.	4. El sistema registra a l'usuari i li envia un correu electrònic notificant-ho.

Casos d'error:
3. El correu electrònic no té un format vàlid. 3. La identificació de l'usuari no ha estat donada d'alta encara.

Cas d'ús:	**Donar d'alta un professor**
Descripció:	L'objectiu és donar d'alta un professor al sistema.
Actors:	Sistema i Administrador
Precondicions:	L'adminsitrador ha fet login a InnovaCampus i indica que vol donar d'alta a un nou professor.
Postcondicions:	El nou professor és donat d'alta i ja es pot registrar.

Flux principal:	
Administrador	**Sistema**
1. L'adminsitrador indica que vol donar d'alta un nou professor.	2. El sistema li proporciona el formulari per omplir amb les dades del nou professor.
3. L'administrador introdueix el nom, cognoms, identificació, telèfon i correu electrònic del nou professor.	4. El sistema dona d'alta el nou professor i ho notifica.

Casos d'error:
3. Alguna de les dades introduïdes pot ser incorrecta. No es guarda la informació del nou professor i s'informa de l'error.

Cas d'ús:	**Donar de baixa un professor**
Descripció:	L'objectiu és donar de baixa un professor del sistema.
Actors:	Sistema i Administrador
Precondicions:	L'administrador ha fet login a InnovaCampus i vol donar de baixa un professor.
Postcondicions:	El professor està donat de baixa.

Flux principal:	
Administrador	**Sistema**
1. L'administrador indica que vol donar de baixa un professor.	2. El sistema generà una llista dels professors.
3. L'administrador selecciona de la llista al professor que vol donar de baixa.	4. El sistema dona de baixa al professor i ho notifica.

Cas d'ús:	**Donar d'alta un anunci**
Descripció:	L'objectiu és que l'administrador doni d'alta un nou anunci al sistema.
Actors:	Sistema i Administrador
Precondicions:	L'administrador ha fet login a InnovaCampus i vol escriure un nou anunci.
Postcondicions:	S'ha donat d'alta un nou anunci al sistma.

Flux principal:	
Administrador	**Sistema**
1. L'administrador indica que vol donar d'alta un nou anunci.	2. El sistema demana a l'usuari que elegeixi el destinatari de l'anunci.
3. L'administrador elegeix el destinatari (professors, alumnes, ambdós o un anunci general)	4. El sistema registra la informació i demana a l'usuari el text de l'anunci.
5. L'administrador introdueix el text de l'anunci.	6. El sistema dona d'alta el nou anunci i ho notifica.

Cas d'ús:	**Donar de baixa un anunci**
Descripció:	L'objectiu és que l'administrador pugui donar de baixa un anunci.
Actors:	Sistema i Administrador
Precondicions:	L'administrador a fet login a InnovaCampus i vol donar de baixa un anunci.
Postcondicions:	S'ha donat de baixa un anunci del sistema

Flux principal:	
Administrador	**Sistema**
1. L'administrador indica que vol donar de baixa un anunci.	2. El sistema demana que elegeixi el destinatari de l'anunci que vol eliminar.
3. L'adminsitrador elegeix el destinatari (professors, alumnes, ambdós o un anunci general)	4. El sistema registra la informació i genera una llista amb els anuncis d'aquell destinatari.
5. L'administardor selecciona l'anunci que desitja esborrar	6. El sistema dona de baixa l'aunci i ho notifica.

Cas d'ús:	**Veure anuncis de l'administrador**
Descripció:	L'objectiu és que l'usuari pugui veure els anuncis de l'administrador.
Actors:	Sistema i Professor o Sistema i Alumne
Precondicions:	L'usuari ha fet login a InnovaCampus.
Postcondicions:	L'usuari visualitza els anuncis de l'administrador.

Flux principal:	
Usuari	**Sistema**
1. L'usuari indica que vol veure els anuncis publicats per l'administrador.	2. El sistema genera una llista amb tots els missatges de l'administrador que pot veure aquest usuari.
3. L'usuari visualitza els anuncis.	

Cas d'ús:	**Veure anuncis de l'administrador per a usuaris no autenticats**
Descripció:	L'objectiu és que un usuari no autenticat pugui veure els anuncis generals publicats per l'administrador.
Actors:	Sistema i Usuari no autenticat
Precondicions:	L'usuari no ha fet login a InnovaCampus.
Postcondicions:	L'usuari visualitza els anuncis generals.

Flux principal:	
Usuari no autenticat	**Sistema**
1.L'usuari indica que vol veure els anuncis generals publicats per l'administrador.	2. El sistema genera una llista amb aquets anuncis.
3. L'usuari visualitza els anuncis.	

Cas d'ús:	**Donar d'alta una assignatura**
Descripció:	L'objectiu és que un professor pugui donar d'alta una assignatura nova al sistema.
Actors:	Sistema i Professor
Precondicions:	El professor a fet login a InnovaCampus.
Postcondicions:	S'ha donat d'alta una nova assignatura.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol donar d'alta una nova assignatura	2. El sistema demana les dades de la nova assignatura.
3. El professor introdueix el nom, descripció i codi de la nova assignatura.	4. El sistema dona d'alta la nova assignatura.

Casos d'error:
El codi de la nova assignatura ja existeix o no és vàlid. No es dóna d'alta la nova assignatura i s'informa de l'error.

Cas d'ús:	**Donar de baixa una assignatura**
Descripció:	L'objectiu és que el professor pugui donar de baixa una assignatura. Això implica que s'esborrin tots els temes, objectius, preguntes, respotes, tests i resultats que estiguin associats a aquesta assignatura.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i vol donar de baixa una assignatura
Postcondicions:	Es dona de baixa l'assignatura i tota la informació del sistema que te associada.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol donar de baixa una assignatura.	2. El sistema genera de les assignatures del professor.
3. El professor escull una de les assignatures de la llista.	4. El sistema dóna de baixa l'assignatura i tota la informació assoicada.

Cas d'ús:	**Modificar dades d'una assignatura**
Descripció:	L'objectiu és que el professor pugui modificar les dades d'una assignatura.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i vol modificar les dades d'una assignatura.
Postcondicions:	S'han modificat les dades de l'assignatura.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol modificar les dades d'una assignatura.	2. El sistema genera una llista de les assignatures del professor.
3. El professor escull una de les assignatures	4. El sistema mostra la informació actual de l'assignatura.
5. El professor fa els canvis oportuns.	6. El sistema guarda les modificacions.

Casos d'error:
5. Alguna de les dades introduïdes no és correcta. No es guarden els canvis i s'informa de l'error.

Cas d'ús:	**Reiniciar una assignatura**
Descripció:	L'objectiu és que el professor pugui reiniciar una assignatura, desmatriculant a tots els estudiants i esborran els resultats dels tests dels alumnes.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus
Postcondicions:	S'ha reiniciat l'assignatura.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol reiniciar una assignatura.	2. El sistema genera una llista de les assignatures del professor.
3. El professor escull l'assignatura.	4. El sistema reinicia l'assignatura.

Cas d'ús:	**Donar d'alta un professor a una assignatura**
Descripció:	L'objectiu és que un professor pugui donar d'alta a una assignatura a un altre professor.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i vol donar d'alta a una a professor d'una assignatura. Aquest professor ha d'haver estat donat d'alta a l'aplicació.
Postcondicions:	El professor ha estat donant d'alta a l'assignatura.

Flux principal:	
Professor	**Sistema**
1. El professor accedeix a la zona de l'assigntaura.	
2. El professor indica que vol donar d'alta un professor a l'assignatura.	3. El sistema genera una llista amb els possibles professors.
4. El professor escull a un professor de la llista.	5. El sistema dóna d'alta el professor escollit a l'assignatura.

Cas d'ús:	**Donar d'alta un alumne a una assignatura**
Descripció:	L'objectiu és que un professor pugui donar d'alta un alumne a una assignatura, que el pugui matricular.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i vol matricular a un alumne.
Postcondicions:	El alumne està matriculat de l'assignatura.

Flux principal:	
Professor	**Sistema**
1. El professor accedeix a la zona de l'assignatura.	
2. El professor indica que vol matricular a un alumne.	3. El sistema li demana les dades de l'estudiant.
4. El professor introdueix el nom, cognoms i identificador de l'alumne.	5. El sistema matricula l'alumne a l'assignatura.

Casos d'error:
4. El identificador de l'alumne no és vàlid. L'alumne no es matricula i s'informa de l'error. mitjançant

Cas d'ús:	**Donar de baixa un professor d'una assignatura**
Descripció:	L'objectiu és que un professor pugui donar de baixa un altre professor d'una assignatura.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i vol donar de baixa a una a professor d'una assigntaura.
Postcondicions:	El professor ha estat donant de baixa de l'assignatura.

Flux principal:	
Professor	**Sistema**
1. El professor accedeix a la zona de l'assigntaura.	
2. El professor indica que vol donar de baixa un professor de l'assignatura.	3. El sistema genera una llista amb els possibles professors.
4. El professor escull a un professor de la llista.	5. El sistema dóna de baixa el professor escollit a l'assignatura.

Cas d'ús:	**Donar d'alta una llista d'alumnes a l'assignatura mitjançant un fitxer**
Descripció:	L'objectiu és que un professor pugui matricular una llista d'alumnes a una assignatura a partir d'un fitxer.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i vol matricular un grup d'alumnes mitjançant un fitxer.
Postcondicions:	S'ha matriculat una llista d'alumnes a una assignatura mitjançant un fitxer.

Flux principal:	
Professor	**Sistema**
1. El professor accedeix a la zona de l'assignatura.	
2. El professor indica que vol matricular una llista d'alumnes mitjançant un fitxer.	3. El sistema li demana el fitxer.
4. El professor selecciona el fitxer.	El sistema matricula la llista d'alumnes a l'assignatura.

Casos d'error:
4. La ruta del fitxer introduïda no és correcta o el fitxer no existeix. S'informa de l'error. 4. El format del fitxer no és el correcte. S'informa de l'error.

Cas d'ús:	**Donar de baixa un alumne de l'assignatura**
Descripció:	L'objectiu és que un professor pugui donar de baixa un alumne d'una assignatura.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i l'alumne està matriculat a l'assignatura.
Postcondicions:	L'alumne ja no està matriculat de l'assignatura.

Flux principal:	
Professor	**Sistema**
1. El professor accedeix a la zona de l'assignatura.	
2. El professor indica que vol donar de baixa un alumne de l'assignatura.	3. El sistema li genera una llista dels alumnes matriculats a l'assignatura.
4. El professor selecciona l'alumne que vol donar de baixa.	El sistema dóna de baixa a l'alumne.

Cas d'ús:	**Donar d'alta un anunci al taulell d'anuncis de l'assignatura**
Descripció:	L'objectiu és que el professor pugui donar d'alta un anunci de l'assignatura.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i vol donar d'alta un anunci d'una assignatura.
Postcondicions:	S'ha donat d'alta un nou anunci per l'assignatura.

Flux principal:	
Professor	**Sistema**
1. El professor accedeix a la zona de l'assignatura.	
2. El professor indica que vol donar d'altar un nou anunci per l'assignatura.	3. El sistema li demana el text de l'anunci.
4. El professor introdueix el text de l'anunci.	5. El sistema dóna d'alta el nou anunci de l'assignatura.

Cas d'ús:	**Donar de baixa un anunci del taulell d'anuncis de l'assignatura**
Descripció:	L'objectiu és que el professor pugui donar de baixa un anunci d'una assignatura.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i vol donar de baixa un anunci d'una assignatura.
Postcondicions:	S'ha donat de baixa un anunci de l'assignatura.

Flux principal:	
Professor	**Sistema**
1. El professor accedeix a la zona de l'assignatura.	
2. El professor indica que vol donar d'alta de baixa un anunci de l'assignatura.	3. El sistema genera una llista del anuncis de l'assignatura.
4. El professor selecciona l'anunci que vol esborrar.	El sistema dóna de baixa l'anunci i ho notifica.

Cas d'ús:	**Veure anuncis del taulell d'anuncis d'una assignatura**
Descripció:	L'objectiu és que l'usuari pugui vuere anuncis del taulell d'anuncis de l'assignatura.
Actors:	Sistema i Professor o Sistema i Alumne
Precondicions:	L'usuari ha fet login a InnovaCampus i té accés a l'assignatura.
Postcondicions:	L'usuari pot veure els anuncis de l'assignatura.

Flux principal:	
Usuari	**Sistema**
1. L'usuari accedeix a la zona de l'assignatura.	
2. L'usuari indica que vol veure els anuncis de l'assignatura.	3. El sistema genera una llista amb tots els anuncis de l'assignatura.
4. L'usuari visualitza els anuncis.	

D.2 Casos d'ús de StatisticsManagement

Cas d'ús:	**Consultar estadístiques d'una assignatura**
Descripció:	L'objectiu és que el professor pugui veure les estadístiques d'una assignatura.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i ha entrat dins de la part de l'assignatura.
Postcondicions:	El professor pot visualitzar les estadístiques de l'assignatura.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol veure les estadístiques de l'assignatura.	2. El sistema genera les estadístiques i gràfiques de l'assignatura.
3. El professor visualitza les estadístiques del tema.	

Cas d'ús:	**Consultar estadístiques d'un tema**
Descripció:	L'objectiu és que el professor pugui visualitzar les estadístiques d'un tema.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i ha entrat dins de l'assignatura.
Postcondicions:	El professor pot visualitzar les estadístiques.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol veure les estadístiques d'un tema.	2. El sistema genera una llista dels temes de l'assignatura.
3. El professor escull el tema.	4. El sistema genera les estadístiques i gràfiques del tema.
5. El professor visualitza les estadístiques del tema.	

Cas d'ús:	**Consultar estadístiques d'un objectiu d'un tema**
Descripció:	L'objectiu és que el professor pugui veure les estadísitques d'un objectiu en concret.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus, ha entrat dins de l'assignatura i visualitza les estadístiques d'un tema.
Postcondicions:	El professor pot veure les estadístiques de l'objectiu.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol veure les estadístiques d'un objectiu del tema.	2. El sistema genera una llista amb els objectius del tema.
3. El professor escull l'objectiu que desitja.	4. El sistema genera les estadístiques i gràfiques de l'objectiu.
5. El professor visualitza les estadístiques de l'objectiu.	

Cas d'ús:	**Consultar estadístiques d'una pregunta**
Descripció:	L'objectiu és que el professor pugui veure les estadístiques d'una pregunta.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus, ha entrat dins de l'assignatura, ha visualitzat les estadísitques d'un tema i visualitza les estadístiques d'un objectiu.
Postcondicions:	El professor pot veure les estadístiques d'una pregunta.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol veure les estadístiques d'una pregunta de l'objectiu.	2. El sistema genera una llista amb les preguntes de l'objectiu.
3. El professor escull la pregunta.	4. El sistema genera les estadístiques i obté les respostes de la pregunta.
5. El professor visualitza un informe sobre la pregunta amb la informació general i estadístiques.	

Cas d'ús:	**Consultar estadístiques d'una assignatura d'un estudiant**
Descripció:	L'objectiu és que un professor pugui veure les estadístiques d'un alumne per una assignatura.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus, ha entrat dins de l'assignatura.
Postcondicions:	El professor pot veure les estadístiques de l'estudiant per una assignatura.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol veure les estadístiques d'un estudiant.	2. El sistema genera una llista dels estudiants matriculats a l'assignatura.
3. El professor escull l'estudiant.	4. El sistema genera les estadístiques i gràfiques de l'estudiant per l'assignatura.
5. El professor visualitza les estadístiques de l'estudiant.	

Cas d'ús:	**Consultar estadístiques d'un tema d'un estudiant**
Descripció:	L'objectiu és que un professor pugui veure les estadístiques d'un alumne per un tema.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i ha entrat dins de l'assignatura.
Postcondicions:	El professor pot veure les estadístiques de l'estudiant per un tema.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol veure les estadístiques d'un estudiant.	2. El sistema genera una llista dels estudiants matriculats a l'assignatura.
3. El professor escull l'estudiant.	4. El sistema genera les estadístiques i gràfiques de l'estudiant per l'assignatura.
5. El professor indica que vol veure les estadístiques per temes de l'estudiant.	6. El sistema genera una llista dels temes de l'assginatura.
7. El professor escull el tema.	8. El sistema genera les estadístiques i gràfiques de l'estudiant per aquell tema.
9. El professor visualitza les estadístiques de l'estudiant.	

Cas d'ús:	**Consultar les estadístiques d'un objectiu d'un estudiant.**
Descripció:	L'objectiu és que un professor pugui veure les estadístiques d'un alumne per un objectiu.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i ha entrat dins de l'assignatura.
Postcondicions:	El professor pot veure les estadístiques de l'estudiant per un objectiu.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol veure les estadístiques d'un estudiant.	2. El sistema genera una llista dels estudiants matriculats a l'assignatura.
3. El professor escull l'estudiant.	4. El sistema genera les estadístiques i gràfiques de l'estudiant per l'assignatura.
5. El professor indica que vol veure les estadístiques per temes de l'estudiant.	6. El sistema genera una llista dels temes de l'assginatura.
7. El professor escull el tema. 8. El professor indica que vol veure les estadístiques per objectius de l'estudiant. 10. El professor escull l'objectiu.	9. El sistema genera una llista dels objectius del tema. 11. El sistema genera les estadístiques i gràfiques de l'estudiant per aquell objectiu.
12. El professor visualitza les estadístiques de l'estudiant.	

D.3 Casos d'ús de SubjectManagement

Cas d'ús:	**Donar d'alta un tema a una assignatura**
Descripció:	L'objectiu és que el professor pugui donar d'alta un tema a una assignatura.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCamps i ha entrat dins l'assignatura.
Postcondicions:	S'ha donat d'alta un nou tema a l'assignatura.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol donar d'alta un tema nou.	2. El sistema li demana el títol i la descripció del tema.
3. El professor introdueix les dades sobre el tema.	4. El sistema dóna d'alta el nou tema i ho notifica.

Casos d'error:
3. Les dades introduïdes no són correctes. No es produeix la inserció del tema i s'informa de l'error.

Cas d'ús:	**Donar de baixa un tema d'una assignatura**
Descripció:	L'obectiu és que el professor pugui donar de baixa un tema d'una assignatura. S'ha de tenir en compte que al eliminar un tema, també s'eliminarà la informació relacionada amb aquest tema, com els objectius, preguntes, respostes, resultats i tests.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i ha entrat dins l'assignatura.
Postcondicions:	S'ha esborrat un tema de l'assignatura i tota la informació relacionada amb aquest tema.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol donar de baixa un tema de l'assignatura.	2. El sistema genera una llista amb els temes de l'assignatura.
3. El professor escull el tema de la llista.	4. El sistema esborra tota la informació del tema i ho notifica.

Cas d'ús:	**Modificar un tema d'una assignatura**
Descripció:	L'objectiu és que el professor pugui modificar les dades d'un tema, com el nom i la descripció.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i ha entrat dins l'assignatura.
Postcondicions:	El professor ha modificat les dades del tema.

Flux principal:	
Usuari	**Sistema**
1. El professor indica que vol modificar les dades d'un tema.	2. El sistema genera una llista amb els temes de l'assignatura.
3. El professor escull el tema desitjat.	4. El sistema mostra la informació actual del tema.
5. El professor introdueix les dades que desitja canviar.	6. El sistema guarda les dades i ho notifica.

Casos d'error:
5. Les dades introduïdes no són correctes. No es produeix la modificació del tema i s'informa de l'error.

Cas d'ús:	**Obtenir els temes d'una assignatura**
Descripció:	L'objectiu és que l'usuari pugui veure un llistat dels temes de l'assignatura.
Actors:	Sistema i Professor o Sistema i Alumne
Precondicions:	L'usuari ha fet login a InnovaCampus i ha entrat dins l'assignatura.
Postcondicions:	L'usuari visualitza tots els temes de l'assignatura.

Flux principal:	
Usuari	**Sistema**
1. L'usuari indica que vol veure els temes de l'assignatura.	2. El sistema genera una llista de tots els temes de l'assignatura.
3. L'usuari visualitza tots els temes de l'assignatura.	

Cas d'ús:	**Obtenir un tema**
Descripció:	L'objectiu és que l'usuari pugui veure la informació d'un tema d'una assignatura.
Actors:	Sistema i Professor o Sistema i Alumne
Precondicions:	L'usuari ha fet login a InonvaCampus i ha entrat dins l'assignatura.
Postcondicions:	L'usuari visualitza la informació del tema.

Flux principal:	
Usuari	**Sistema**
1. L'usuari indica que vol veure la informació d'un tema.	2. El sistema genera una llista dels temes de l'assignatura.
3. L'usuari escull un tema	4. El sistema obté la informació d'aquest tema i li mostra per pantalla.

Cas d'ús:	**Donar d'alta un objectiu a un tema.**
Descripció:	L'objectiu és que el professor pugui donar d'alta un objectiu a un tema d'una assignatura.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i ha entrat dins de l'assignatura.
Postcondicions:	S'ha donat d'alta un nou objectiu a un tema de l'assignatura.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol donar d'alta un nou objectiu.	2. El sistema genera una llista amb els temes de l'assignatura.
3. El professor escull el tema.	4. El sistema demana les dades del nou objectiu.
5. El professor introdueix les dades: nom, descripció, nombre de preguntes que tindran els tests...	6. El sistema dóna d'alta el nou objectiu i ho notifica.

Casos d'error:
5. Les dades introduïdes no són correctes. No es produeix la inserció de l'objectiu i s'informa de l'error.

Cas d'ús:	**Donar de baixa un objectiu d'un tema**
Descripció:	L'objectiu és que el professor pugui donar de baixa un objectiu. S'ha de tenir en compte que al eliminar un objectiu, també s'eliminarà la informaicó associada amb l'objectiu, com les preguntes, respostes, resultats i tests.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i ha entrat dins de l'assignatura.
Postcondicions:	S'ha esborrat un objectiu d'un tema i tota la informació relacionada amb aquest objectiu.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol donar de baixa un objectiu.	2. El sistema gerera una llista dels temes de l'assignatura.
3. El professor escull el tema de l'objectiu.	4. El sistema genera una llista dels objectius del tema.
5. El professor escull el objectiu.	6. El sistema esborra tota la informació de l'objectiu i ho notifica.

Cas d'ús:	**Modificar un objectiu d'un tema**
Descripció:	L'objectiu és que el professor pugui modificar les dades d'un objectiu.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i ha entrat dins de l'assignatura.
Postcondicions:	El professor ha modificat les dades de l'objectiu.

Flux principal:	
Usuari	**Sistema**
1. El professor indica que vol modificar les dades d'un objectiu.	2. El sistema genera una llista amb els temes de l'assignatura.
3. El professor escull el tema.	4. El sistema genera una llista amb els objectius del tema.
5. El professor escull l'objectiu.	6. El sistema mostra la informació actual de l'objectiu.
7. El professor introdueix les dades que desitja canviar.	8. El sistema guarda les dades i ho notifica.

Casos d'error:
7. Les dades introduïdes no són correctes. No es produeix la modificació de l'objectiu i s'informa de l'error.

Cas d'ús:	**Obtenir els objectius d'un tema**
Descripció:	L'objectiu és que l'usuari pugui veure un llistat dels objectius d'un tema.
Actors:	Sistema i Professor o Sistema i Alumne
Precondicions:	L'usuari ha fet login a InnovaCampus i ha entrat dins l'assignatura.
Postcondicions:	L'usuari visualitza tots els objectius d'un tema.

Flux principal:	
Usuari	**Sistema**
1. L'usauri indica que vol veure els objectius d'un tema.	2. El sistema genera una llista amb tots els temes de l'assignatura.
3. L'usuari escull el tema.	4. El sistema genera una llista amb tots els objectius del tema.
5. L'usuari visualitza tots els objectius del tema.	

Cas d'ús:	**Obtenir un objectiu**
Descripció:	L'objectiu és que l'usuari pugui veure la informació d'un objectiu.
Actors:	Sistema i Professor o Sistema i Alumne
Precondicions:	L'usuari ha fet login a InnovaCampus i ha entrat dins de l'assignatura.
Postcondicions:	L'usuari visualitza la informació de l'objectiu.

Flux principal:	
Usuari	**Sistema**
1. L'usauri indica que vol veure la informació d'un objectiu.	2. El sistema genera una llista amb tots els temes de l'assignatura.
3. L'usuari escull el tema.	4. El sistema genera una llista amb tots els objectius del tema.
5. L'usuri escull l'objectiu.	6. El sistema obté la informació d'aquest objectiu i li mostra per pantalla.

Cas d'ús:	**Donar d'alta un recurs a un objectiu**
Descripció:	L'objectiu és que el professor pugui donar d'alta un recurs a un recurs.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i ha entrat dins l'assignatura.
Postcondicions:	S'ha afegit un nou recurs a l'objectiu.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol afegir un nou recurs a un objectiu.	2. El sistema genera una llista de tots els temes de l'assignatura.
3. El professor escull el tema.	4. El sistema genera una llista de tots els objectius del tema.
5. El professor escull l'objectiu	6. El sistema li demana les dades del recurs.
7. El professor introdueix el tipus de fitxer, el nom i el fitxer.	8. El sistema dóna d'alta el recurs i ho notifica.

Casos d'error:
7. Les dades introduïdes no són correctes o el fitxer no existeix. No es dóna d'alta el recurs i s'informa de l'error.

Cas d'ús:	**Donar de baixa un recurs d'un objectiu**
Descripció:	L'objectiu és que el professor pugui donar de baixa un recurs d'un objectiu. També s'eliminaran les preguntes, respostes i tests associats a aquest recurs.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i ha entrat dins l'assignatura.
Postcondicions:	S'ha donat de baixa un recurs de l'objectiu.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol donar de baixa un recurs d'un objectiu.	2. El sistema genera una llista de tots els temes de l'assignatura.
3. El professor escull el tema.	4. El sistema genera una llista de tots els objectius del tema.
5. El professor escull l'objectiu	6. El sistema genera una llista amb tots els recursos de l'objectiu.
7. El professor escull el recurs.	8. El sistema dóna de baixa el recurs i ho notifica.

Cas d'ús:	**Modificar un recurs d'un objectiu**
Descripció:	L'objectiu és que el professor pugui modificar les dades d'un recurs.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i ha entrat dins l'assignatura.
Postcondicions:	S'han modificat les dades del recurs.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol modificar les dades d'un recurs.	2. El sistema genera una llista de tots els temes de l'assignatura.
3. El professor escull el tema.	4. El sistema genera una llista de tots els objectius del tema.
5. El professor escull l'objectiu	6. El sistema genera una llista amb tots els recursos de l'objectiu.
7. El professor escull el recurs.	8. El sistema mostra la informació actual del recurs.
9. El professor introdueix les dades que desitja canviar.	10. El sistema guarda les dades i ho notifica.

Casos d'error:
7. Les dades introduïdes no són correctes. No es produeix la modificació del recurs i s'informa de l'error.

Cas d'ús:	**Obtenir els recursos d'un objectiu**
Descripció:	L'objectiu és que el professor pugui veure un llistat dels recursos d'un objectiu.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i ha entrat dins l'assignatura.
Postcondicions:	El professor visualitza tots els recursos d'un objectiu.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol veure tots els recursos d'un objectiu.	2. El sistema genera una llista de tots els temes de l'assignatura.
3. El professor escull el tema.	4. El sistema genera una llista de tots els objectius del tema.
5. El professor escull l'objectiu	6. El sistema genera una llista amb tots els recursos de l'objectiu.
7. El professor visualitza tots els recursos de l'objectiu.	

Cas d'ús:	**Visualitzar un recurs**
Descripció:	L'objectiu és que el professor pugui veure la informació un recurs.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i ha entrat dins l'assignatura.
Postcondicions:	El professor visualitza la informació d'un recurs.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol veure la informació d'un recurs.	2. El sistema genera una llista de tots els temes de l'assignatura.
3. El professor escull el tema.	4. El sistema genera una llista de tots els objectius del tema.
5. El professor escull l'objectiu	6. El sistema genera una llista amb tots els recursos de l'objectiu.
7. El professor escull el recurs.	8. El sistema obté la informació del recurs i li mostra per pantalla.

D.4 Casos d'ús de TestManagement

Cas d'ús:	**Donar d'alta una pregunta a un objectiu**
Descripció:	L'objectiu és que le professor pugui donar d'alta una pregunta dins d'un objectiu.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i ha entrat dins l'assignatura.
Postcondicions:	S'ha donat d'alta una nova pregunta d'un objectiu.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol donar d'alta una nova pregunta.	2. El sistema genera una llista dels temes de l'assignatura.
3. El professor escull el tema.	4. El sistema genera una llista dels objectius del tema.
5. El professor escull l'objectiu	6. El sistema li demana les dades de la pregunta.
7. El professor introdueix el text de la pregunta, les respotes i si vol associar un recurs o establir dependència entre preguntes.	8. El sistema dóna d'alta la pregunta i ho notifica.

Casos d'error:
7. Les dades introduïdes no són correctes. No es dóna d'alta la pregunta i s'informa de l'error.

Cas d'ús:	**Inhabilitar una pregunta d'un objectiu**
Descripció:	L'objectiu és que el professor pugui inhabilitar una pregunta. D'aquesta manera es poden conservar les estadístiques d'aquesta pregunta, però ja no sortirà més a cap test.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i ha entrat dins l'assignatura.
Postcondicions:	S'ha inhabilitat la pregunta.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol inhabilitar una pregunta.	2. El sistema genera una llista dels temes de l'assignatura.
3. El professor escull el tema.	4. El sistema genera una llista dels objectius del tema.
5. El professor escull l'objectiu	6. El sistema genera una llista amb les preguntes de l'objectiu.
7. El professor escull la pregunta.	8. El sistema inhabilita la pregunta i ho notifica.

Cas d'ús:	**Donar de baixa una pregunta d'un objectiu**
Descripció:	L'objectiu és que el professor pugui donar de baixa una pregunta d'un objectiu. També s'eliminaran les respotes, resultats i tests associats a la pregunta.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i ha entrat dins l'assignatura.
Postcondicions:	S'ha donat de baixa una pregunta i tota la informació associada.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol donar de baixa una pregunta.	2. El sistema genera una llista dels temes de l'assignatura.
3. El professor escull el tema.	4. El sistema genera una llista dels objectius del tema.
5. El professor escull l'objectiu	6. El sistema genera una llista amb les preguntes de l'objectiu.
7. El professor escull la pregunta.	8. El sistema dóna de baixa la pregunta i tota la informació associada i ho notifica.

Cas d'ús:	**Modificar una pregunta d'un objectiu**
Descripció:	L'objectiu és que el professor pugui modificar les dades d'una pregunta d'un objectiu.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i ha entrat dins de l'assignatura.
Postcondicions:	S'han modificat les dades de la pregunta.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol modificar les dades d'una pregunta.	2. El sistema genera una llista dels temes de l'assignatura.
3. El professor escull el tema.	4. El sistema genera una llista dels objectius del tema.
5. El professor escull l'objectiu	6. El sistema genera una llista amb les preguntes de l'objectiu.
7. El professor escull la pregunta.	8. El sistema mostra la informació actual de la pregunta.
9. El professor introdueix les noves dades.	10. El sistema guarda les dades i ho notifica.

Casos d'error:
9. Les dades introduïdes no són correctes. No es produeix la modificació de la pregunta i s'informa de l'error.

Cas d'ús:	**Obtenir totes les preguntes d'un objectiu**
Descripció:	L'objectiu és que el professor pugui veure un llistat de totes les preguntes d'un objectiu.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i ha entrat dins de l'assignatura.
Postcondicions:	El professor visualitza totes les preguntes de l'objectiu.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol veure totes les preguntes d'un objectiu.	2. El sistema genera una llista dels temes de l'assignatura.
3. El professor escull el tema.	4. El sistema genera una llista dels objectius del tema.
5. El professor escull l'objectiu	6. El sistema genera una llista amb les preguntes de l'objectiu.
7. El professor visualitza totes les preguntes de l'objectiu.	

Cas d'ús:	**Visualitzar una pregunta**
Descripció:	L'objectiu és que el professor pugui veure la informació d'una pregunta.
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCamus i ha entrat dins de l'assignatura.
Postcondicions:	El professor visualitza la informació de la pregunta.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol veure la informació d'una pregunta.	2. El sistema genera una llista dels temes de l'assignatura.
3. El professor escull el tema.	4. El sistema genera una llista dels objectius del tema.
5. El professor escull l'objectiu	6. El sistema genera una llista amb les preguntes de l'objectiu.
7. El professor escull la pregunta.	8. El sistema obté la informació de la pregunta i li mostra per pantalla.

Cas d'ús:	**Obtenir totes les preguntes entre un rang de dificultat**
Descripció:	L'objectiu és que el professor pugui veure totes les preguntes entre un rang de dificultats (fàcil, nivell mig o difícil).
Actors:	Sistema i Professor
Precondicions:	El professor ha fet login a InnovaCampus i ha entrat dins de l'assignatura.
Postcondicions:	El professor visualita un llistat de preguntes que pertanyen a un rang de dificultat.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol veure totes les preguntes compreses entre un rang de dificultat.	2. El sistema genera una llista dels temes de l'assignatura.
3. El professor escell el tema.	4. El sistema genera una llista dels objectius del tema.
5. El professor escull l'objectiu	6. El sistema li demana el rang de dificultat.
7. El professor indica quin és el rang de dificultat que vol.	8. El sistema genera una llista amb les preguntes compreses entre aquest rang de dificultat.
9. El professor visualitza les preguntes compreses entre un rang de dificultat.	

Cas d'ús:	**Donar d'alta una resposta a una pregunta**
Descripció:	L'objectiu és que el professor pugui afegir una resposta a una pregunta determinada.
Actors:	Sistema i Professor
Precondicions:	El professor es troba en la edició de la pregunta, tant pot ser en el moment de crear-la com en el de modificar-la.
Postcondicions:	La resposta s'ha inclòs a la pregunta.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol donar d'alta una resposta a una pregunta.	2. El sistema li demana la informació de la resposta.
3. El professor introdueix el nom i el comentari de la resposta i diu si és correcta o no.	4. El sistema dóna d'alta la nova resposta i ho notifica.

Casos d'error:
Les dades introduïdes no són correctes. No es produeix la inserció de la resposta i s'informa de l'error.

Cas d'ús:	**Inhabilitar una resposta d'una pregunta**
Descripció:	L'obejctiu és que el professor pugui inhabilitar una resposta. Així no cal esborrar els resultats i els tests associats a aquesta resposta.
Actors:	Sistema i Professor
Precondicions:	El professor es troba en la modificació de la pregunta.
Postcondicions:	La resposta s'ha inhabilitat.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol inhabilitar una resposta.	2. El sistema li genera una llista amb totes les respostes de la pregunta.
3. El professor escull la resposta.	4. El sistema inhabilita la resposta i ho notifica.

Cas d'ús:	**Donar de baixa una resposta d'una pregunta**
Descripció:	L'objectiu és que el professor pugui donar de baixa una resposta d'una pregunta. També s'eliminaran els resultats i els tests associats a aquesta resposta.
Actors:	Sistema i Professor
Precondicions:	El professor es troba en la modificació de la pregunta.
Postcondicions:	S'ha donat de baixa la resposta i tota la informació associada.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol donar de baixa una resposta.	2. El sistema li genera una llista amb totes les respostes de la pregunta.
3. El professor escull la resposta.	4. El sistema dóna de baixa la resposta i ho notifica.

Cas d'ús:	**Modificar una resposta d'una pregunta**
Descripció:	L'objectiu és que el professor pugui modificar les dades d'una resposta.
Actors:	Sistema i Professor
Precondicions:	El professor es troba en la modificació de la pregunta.
Postcondicions:	S'han modificat les dades de la resposta.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol modificar les dades d'una resposta.	2. El sistema li genera una llista amb totes les respostes de la pregunta.
3. El professor escull la resposta.	4. El sistema li mostra la informació actual de la resposta.
5. El professor introdueix les noves dades.	6. El sistema guarda els canvis i ho notifica.

Casos d'error:
5. Les dades introduïdes no són correctes. No es produeix la modificació de la pregunta i s'informa de l'error.

Cas d'ús:	**Obtenir totes les respostes d'una pregunta**
Descripció:	L'objectiu és que el professor pugui veure un llistat de totes les respostes d'una pregutna.
Actors:	Sistema i Professor
Precondicions:	El professor es troba en la edició de la pregunta.
Postcondicions:	El professor visualitza totes les respotes de la pregunta.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol veure totes les respostes d'una pregunta.	2. El sistema li genera una llista amb totes les respostes de la pregunta.
3. El professor visualitza totes les respostes de la pregunta.	

Cas d'ús:	**Obtenir una resposta**
Descripció:	L'objectiu és que el professor pugui veure la informació d'una resposta.
Actors:	Sistema i Professor
Precondicions:	El professor es troba en la edició de la pregunta.
Postcondicions:	El professor visualitza la informació de la resposta.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol veure la informació d'una resposta.	2. El sistema li genera una llista amb totes les respostes de la pregunta.
3. El professor escull la resposta.	4. El sistema obté la informació de la resposta i li mostra per pantalla.

Cas d'ús:	**Donar d'alta una dependència a una pregunta**
Descripció:	L'objectiu és que el professor pugui afegir una dependència a una determinada pregunta.
Actors:	Sistema i Professor
Precondicions:	El professor es troba en la edició de la pregunta.
Postcondicions:	S'ha establert una nova dependència entre preguntes.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol afegir dependència.	2. El sistema genera una llista amb le possibles preguntes.
3. El professor selecciona les preguntes amb les que vol establir la dependència.	4. El sistema guarda la informació i ho notifica.

Cas d'ús:	**Donar de baixa una dependència a una pregunta**
Descripció:	L'objectiu és que el professor pugui esborrar una dependència entre preguntes.
Actors:	Sistema i Professor
Precondicions:	El professor es troba en la edició de la pregunta.
Postcondicions:	La dependència s'ha eliminat de la pregunta.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol esborrar una dependència de la pregunta.	2. El sistema li mostra les preguntes amb les que depèn.
3. El professor desselecciona la dependència a elminiar.	4. El sistema esborra la dependència i ho notifica.

Cas d'ús:	**Obtenir totes les preguntes que han d'anar abans d'una determinada pregunta**
Descripció:	L'objeciu és que el professor pugui veure un llistat de les preguntes de les quals en depèn una determinada pregunta.
Actors:	Sistema i Professor
Precondicions:	El professor es troba en la edició de la pregunta.
Postcondicions:	El professor visualitza les preguntes de les quals en depèn la pregunta.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol veure les preguntes de les qual en depèn la pregunta que està editant.	2. El sistema genera la llista de les preguntes.
3. El professor visualitza la llista de les preguntes.	

Cas d'ús:	**Obtenir totes les preguntes que han d'anar després d'una determinada pregunta**
Descripció:	L'objeciu és que el professor pugui veure un llistat de les preguntes que depenen d'una determinada pregunta.
Actors:	Sistema i Professor
Precondicions:	El professor es troba en la edició de la pregunta.
Postcondicions:	El professor visualitza les preguntes que depenen de la pregunta.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol veure les preguntes que depenen de la pregunta que està editant.	2. El sistema genera la llista de les preguntes.
3. El professor visualitza la llista de les preguntes.	

Cas d'ús:	**Donar d'alta un recurs a una pregunta**
Descripció:	L'objectiu és que el professor pugui relacionar una pregunta amb un recurs.
Actors:	Sistema i Professor
Precondicions:	El professor es troba en la edició de la pregunta i el recurs ja està donat d'alta en el mateix objectiu que la pregunta.
Postcondicions:	El recurs està relacionat amb la pregunta.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol relacionar un recurs a la pregunta.	2. El sistema genera una llista amb els tipus de rercurs.
3. El professor escull el tipus de recurs.	4. El sistema genera una llista amb els possibles recursos.
5. El professor escull el recurs.	6. El sistema estableix la relació i ho notifica.

Casos d'error:
5. El professor no pot escollir cap recurs perquè no hi ha cap recurs d'aquest tipus donat d'alta i s'informa de l'error.

Cas d'ús:	**Donar de baixa un recurs d'una pregunta**
Descripció:	L'objectiu és que el professor pugui eliminar la relació entre un recurs i una pregunta.
Actors:	Sistema i Professor
Precondicions:	El professor es troba en la edició de la pregunta i la pregunta té un recurs associat..
Postcondicions:	La pregunta ja no té cap recurs associat.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol eliminar la relació del recurs a la pregunta.	2. El sistema esborra la relació i ho notifica.

Cas d'ús:	**Obtenir el recurs d'una pregunta**
Descripció:	L'objectiu és que el professor pugui veure la informació d'un recurs associat a una pregunta.
Actors:	Sistema i Professor
Precondicions:	El professor es troba en la edició de la pregunta i la pregunta té un recurs associat.
Postcondicions:	El professor visualitza la informació del recurs associat a la pregunta.

Flux principal:	
Professor	**Sistema**
1. El professor indica que vol veure la informació del recurs associat a la pregunta.	2. El sistema obté la informació i la mostra per pantalla.

Cas d'ús:	**Generar un test**
Descripció:	L'objectiu és que es generi un test perquè l'alumne el pugui resoldre.
Actors:	Sistema i Alumne
Precondicions:	L'alumne ha fet login a InnovaCampus i ha entrat dins d'una assignatura.
Postcondicions:	S'ha generat un test que l'alumne pot començar a fer.

Flux principal:	
Alumne	**Sistema**
1. L'alumne indica que vol realitzar un test.	2. El sistema li mostra els tipus de test.
3. L'alumne selecciona un tipus de test	4. El sistema li genera un test i li mostra per pantalla.

Cas d'ús:	**Contestar les preguntes d'un test**
Descripció:	L'objectiu és que l'alumne faci un test en un determinat temps. Va escollint les respostes que creu correctes de les diferents preguntes fins a complertar-lo. Al final se li mostra un informe sobre el test amb la puntuació i les respostes escollides.
Actors:	Sistema i Alumne
Precondicions:	Se li ha generat un test a l'alumne per a que el pugui contestar.
Postcondicions:	L'alumne ha finalitzat el test i se li mostra un informe amb els resultats.

Flux principal:	
Alumne	**Sistema**
	1. El sistema ha generat un test per l'alumne.
2. L'alumne respon les preguntes del test i el dóna per finalitzat.	3. El sistema guarda la informació i genera un informe.
4. L'alumne visualitza l'informe.	

Cas d'ús:	**Obtenir resultat del test**
Descripció:	L'objectiu és que l'alumne pugui veure el resultat d'un test.
Actors:	Sistema i Alumne
Precondicions:	L'alumne ha fet login a InnovaCampus i ha entart dins l'assignatura.
Postcondicions:	L'alumne visualitza el resultat d'un test.

Flux principal:	
Alumne	**Sistema**
1. L'alumne indica que vol veure el resultat d'un test.	2. El sistema genera una llista amb elst test de l'alumne.
3. L'alumne escull el test.	4. El sistema obté els resultats del test i li mostra per pantalla.

Bibliografia

[1] Marcel·lí Alet Alís. *InnovaCampus: Una aplicació Web d'ajut a l'autoavaluació en una titulació universitària.* 2006.

[2] Xavier Aiguabella Guilera. *Disseny i implementació del mòdul de generació de tests d'una aplicació web d'autoavaluació desenvolupada en comunitat.* 2007.

[3] Ramon Xuriguera Albareda. *Disseny i implementació d'un mòdul d'estadístiques per a una aplicació d'autoavaluació desenvolupada en comunitat* 2007.

[4] Toni Granollers i Saltiveri, Jesús Lorés Vidal, and José Juan Cañas Delgado. *Diseño de sistemas interactivos centrados en el usuario.* Editorial UOC, 2005.

[5] Craig Walls. *Spring.* Ediciones ANAYA MULTIMEDIA, 2008.

[6] Nicholas C. Zakas, Jeremy McPeak, i Joe Fawcett. *Professional AJAX.* Wiley Publishing, Inc., 2006.

[7] Ajax JSP Tag Library. http://ajaxtags.sourceforge.net/

[8] La Catedral y el bazar de Eric S. Raymond. http://biblioweb.sindominio.net/telematica/catedral.html

[9] Cewolf http://cewolf.sourceforge.net/

[10] Hojas de estilo en cascada. http://es.wikipedia.org/wiki/Hojas_de_estilo_en_cascada

[11] Internationalization and localization. http://en.wikipedia.org/wiki/Internationalization_and_localization

[12] IEC 1131 - PROGRAMMABLE CONTROLLERS. Part 7 - Fuzzy Control Programming. http://jfuzzylogic.sourceforge.net/doc/iec_1131_7_cd1.pdf

[13] The Source for Java Developers. http://java.sun.com/

[14] The Java Tutorials. http://java.sun.com/docs/books/tutorial/

[15] JFreeChart. http://www.jfree.org/jfreechart/

[16] jFuzzyLogic. http://jfuzzylogic.sourceforge.net/

[17] The Java Persistence API. http://java.sun.com/developer/technicalArticles/J2EE/jpa/

[18] JUnit. http://en.wikipedia.org/wiki/JUnit

[19] JUnit.org Resources for Test Driven Development. http://www.junit.org/

[20] Lògica difusa. http://en.wikipedia.org/wiki/Fuzzy_logic

[21] MD5. http://en.wikipedia.org/wiki/MD5

[22] SHA hash functions. http://en.wikipedia.org/wiki/SHA_hash_functions

[23] Spring. http://www.springsource.org/

[24] Spring Documentation. http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/

[25] Spring per aplicacions .NET. http://www.springframework.net/

[26] Spring Security. http://static.springsource.org/spring-security/site/

[27] Subversion. http://subversion.tigris.org/

[28] Apache Tiles. http://tiles.apache.org/

[29] Wang Xiaoyun. http://en.wikipedia.org/wiki/Xiaoyun_Wang